Exemplar work

GCE Computing COMP4

Candidate 6

Please note: every effort has been made to respect the right of anonymity of the source of this project and of those who have contributed to it, without compromising the work or invalidating it for the purpose to which it is to be put. Some personal details of contributors may therefore remain. We ask the reader to respect their right to privacy.

A2 Computing Coursework

Transformations of the graphs of functions

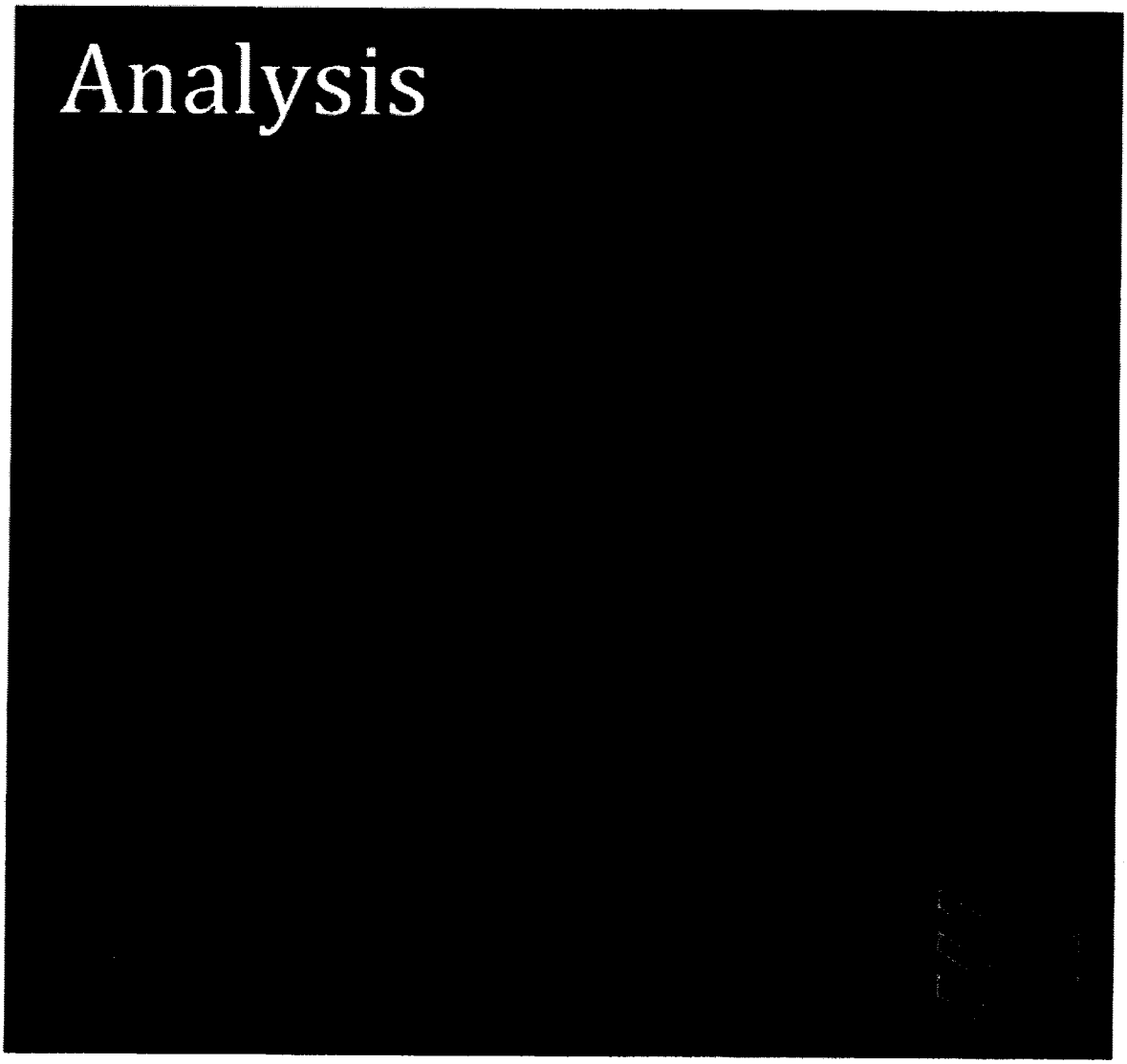
2009/2010

[redacted]

Note:

The page numbers in the separate document, *Commentary on Project 6*, refer to the page numbers given in the bottom right-hand corner of this Exemplar Project 6.

Analysis



Specification Excerpt:

- Evaluate the possible need for development of a computer-based solution to a problem
- Judge the feasibility of a computer-based solution to a problem
- Derive the user, data and processing requirements of a system including a consideration of the human aspects and physical environment
- Specify and document the data and processing requirements for a computer-based solution to the problem

ANALYSIS CONTENTS

Explanation of terminology involved 3

The different types of function transformations and their graphs 4

Functions 8

1. INTRODUCTION 11

1.1 Background 11

1.2 Problem definition 12

2. INVESTIGATION OF OLD SYSTEM 13

2.1 The current System 13

2.2 Data flow diagram of exsisting system 14

2.3 Discussion of problems with the current system 16

3. INVESTIGATION for the NEW SYSTEM 17

3.1 Identication of the prospective users 17

3.2 identification of the user Requirements and acceptable limitations 17

3.3 Data sources, stores and destinations for the proposed system 18

3.4 Data volumes 20

3.5 Data flow diagram of proposed new system 22

3.6 Description of processes and data flows 24

3.7 Analysis Data dictionary 25

4. OBJECTIVES FOR THE PROPOSED SYSTEM 26

4.1 Input, processing and output requirements 26

4.2 performance requirements 26

5. OBJECT ANALYSIS DIAGRAM 27

Inheritance diagram 27

6. REALISTIC appraisal of the feasibility of potential solutions 27

7. JUSTIFICATION of chosen solution 31

EXPLANATION OF TERMINOLOGY INVOLVED

Since my project is going to involve a lot of Mathematical vocabulary, I thought it would be important to define the terms I will use first. The source of the following information is from my A-level Mathematics Textbook: Introducing Pure Mathematics (Smedley & Wiseman, 2001).

Definitions

"Function of x" or "$f(x)$": This is any operation/expression that does something or transforms the variable "x". Throughout the project, instead of saying, "functions of x," I will continually refer to them as just "functions."

A "Transformation" of the graph of a function: The alteration of a graphical plot of a function by changing the actual function itself. E.g. changing the function "(x^2)" to "$(x^2 + 1)$" will change the shape its graph.

Cartesian Coordinates: The usual system for identifying the location of a point in two dimensions. The position of a point in a plane can be represented by a pair of numbers (x, y), and these are collectively called Cartesian Coordinates. E.g. (5,6), (19,-2).

Scale Factor: A numerical value that is used to describe the extent of a graph enlargement. If the magnitude it is positive it means the graph has been made larger but if negative, the graph has been made smaller.

'Compound' function transformation: A combination of two or more function transformations.

($x \in \mathbb{R}$): This means the variable, "x," can take any Real number value (i.e. any decimal or whole number).

Since my project will be referring to the topic of, "The transformations of the graphs of functions," I feel it is also necessary to describe the different types of transformations that the A-level specification includes, and therefore the ones I will cover. In addition, I will include information of the different functions I will use and their untransformed graphs. I will do so under the next two headings:

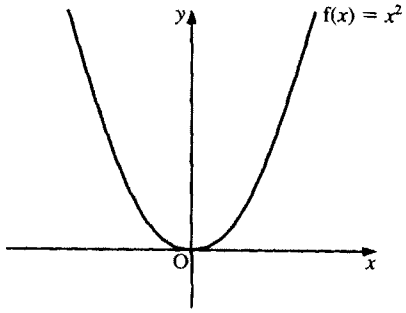
Translations parallel to the y-axis

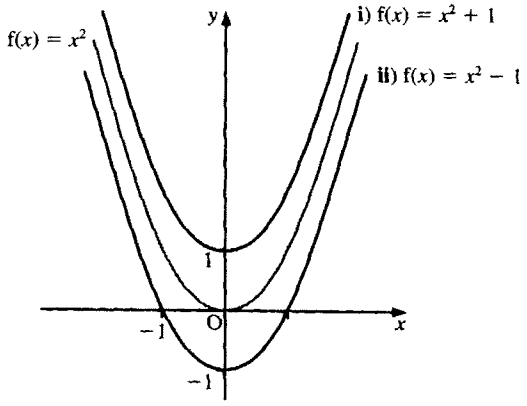
Consider the function f defined by $f(x) = x^2$, $x \in \mathbb{R}$. Plotting the graph of f gives the curve shown on the first of the two diagrams below.

If we now simplify expressions for (i) $f(x) + 1$ and (ii) $f(x) - 1$, we have

$$\text{i) } f(x) + 1 = x^2 + 1 \quad \text{and} \quad \text{ii) } f(x) - 1 = x^2 - 1$$

Plotting graphs of both (i) and (ii) on the same set of axes gives the curves shown on the second of the two diagrams below.





In case (i), the graph of f has been translated 1 unit parallel to the y-axis.

In case (ii), the graph of f has been translated -1 unit parallel to the y-axis.

In general:

- The algebraic transformation $f(x) + a$, where a is a constant, causes a geometric transformation of the graph of f, namely a **translation of a units parallel to the y-axis**.
- The algebraic transformation $f(x) - a$, where a is a constant, causes a geometric transformation of the graph of f, namely a **translation of $-a$ units parallel to the y-axis**.

Translations parallel to the x-axis

Consider the function f defined by $f(x) = 4x - 1,\ x \in \mathbb{R}$.

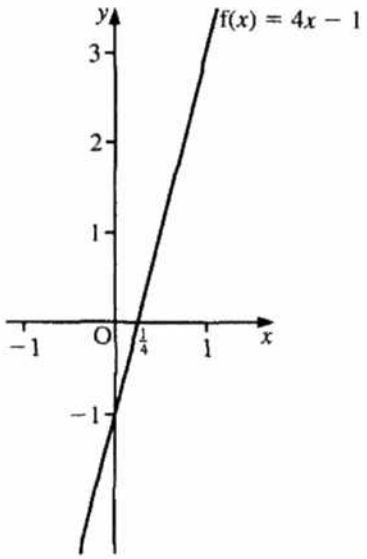
Plotting the graph of f gives the line shown on the left in the two diagrams below.

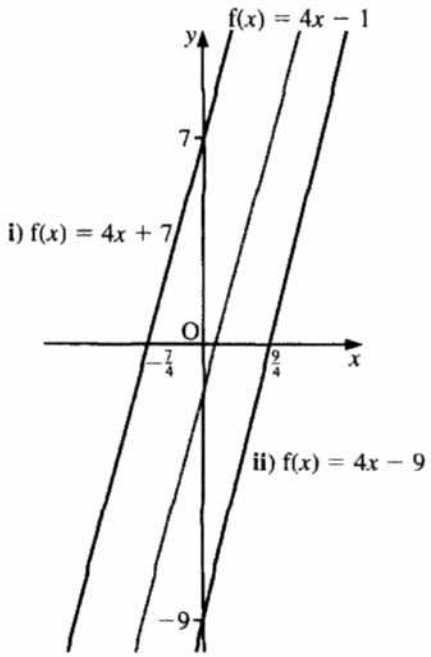
If we now simplify expressions for (i) $f(x + 2)$ and (ii) $f(x - 2)$, we have

i) $f(x + 2) = 4(x + 2) - 1$ and ii) $f(x - 2) = 4(x - 2) - 1$

$= 4x + 7$ $= 4x - 9$

Plotting graphs of both (i) and (ii) on the same set of axes gives the parallel *lines* shown below right.





In case (i), the graph of f has been translated -2 units parallel to the x-axis.

In case (ii), the graph of f has been translated 2 units parallel to the x-axis.

In general:

- The algebraic transformation $f(x + a)$, where a is a constant, causes a geometric transformation of the graph of f, namely a **translation of $-a$ units parallel to the x-axis**.
- The algebraic transformation $f(x - a)$, where a is a constant, causes a geometric transformation of the graph of f, namely a **translation of a units parallel to the x-axis**.

Reflection transformations

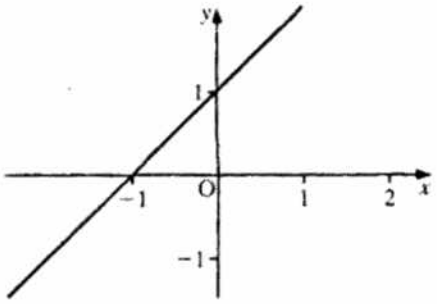
Consider the function f defined by $f(x) = x + 1,\ x \in \mathbb{R}$. Plotting the graph of f gives the line shown below left.

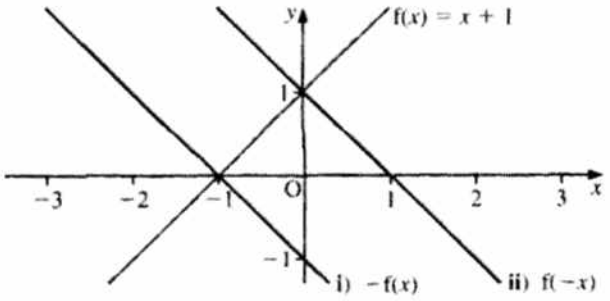
Now consider the functions

i) $-f(x) = -(x + 1) = -x - 1$

ii) $f(-x) = (-x) + 1 = -x + 1$

Plotting graphs of both (i) and (ii) on the same set of axes gives the lines shown below right.





In case (i), the graph of f has been reflected in the x-axis.

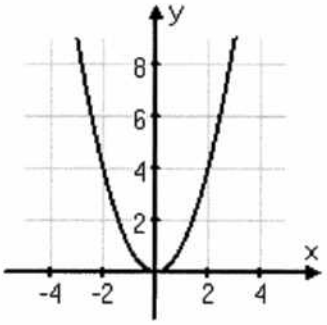
In case (ii), the graph of f has been reflected in the y-axis.

Stretch transformations

Type 1

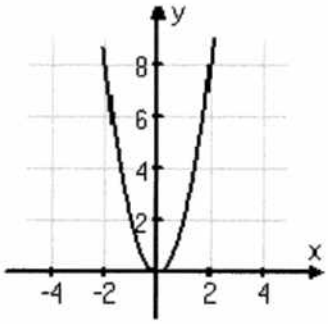
***y*- stretch**

$f(x) = x^2$



The graph of the original, untransformed function

$2f(x) = 2x^2$

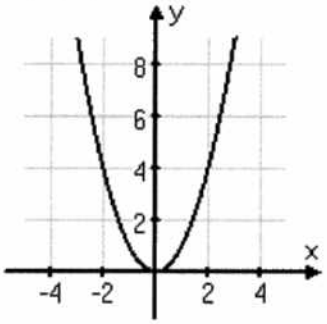


The above transformation stretches the original graph parallel to the y-axis by a scale factor of 2

Type 2

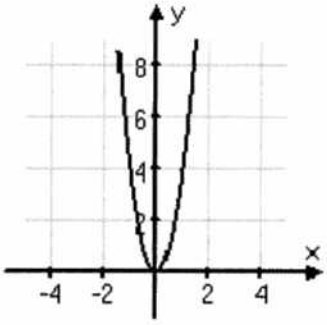
***x*- stretch**

$f(x) = x^2$



The graph of the original, untransformed function

$f(2x) = (2x)^2$



The above transformation stretches the original graph parallel to the x-axis by a scale factor of ½ (i.e. it compresses the graph by a scale factor 2)

In general:

- The algebraic transformation f(ax), where a is a constant, causes a geometric transformation of the graph of f, namely a **stretch parallel to the x-axis by a scale factor of $\frac{1}{a}$.**
- The algebraic transformation a f(x), where a is a constant, causes a geometric transformation of the graph of f, namely a **stretch parallel to the y-axis by a scale factor of a.**

FUNCTIONS

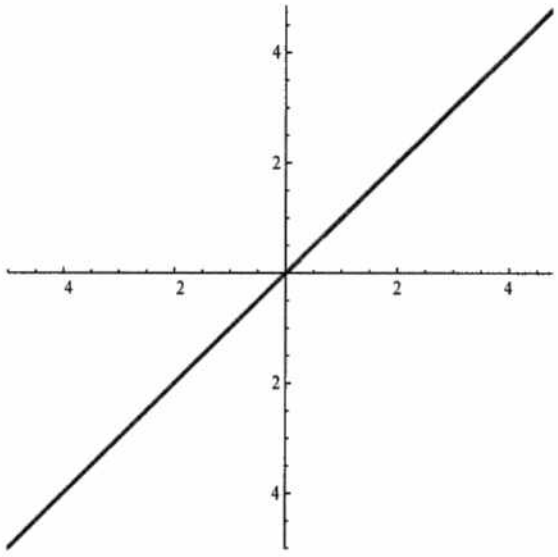
As I said earlier, a:

"Function of x" or "$f(x)$": This is any operation/expression that does something or transforms the variable "x". Throughout the project, instead of saying, "functions of x," I will continually refer to them as just "functions."

I am going to be using nine different mathematical functions. In the following part, I am going to list what they all are and show an example of what each of their graphs look like:

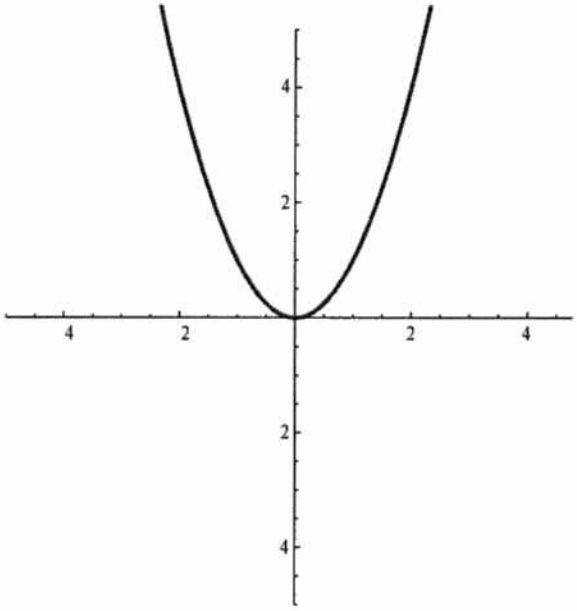
LINEAR

$y = f(x) = x$



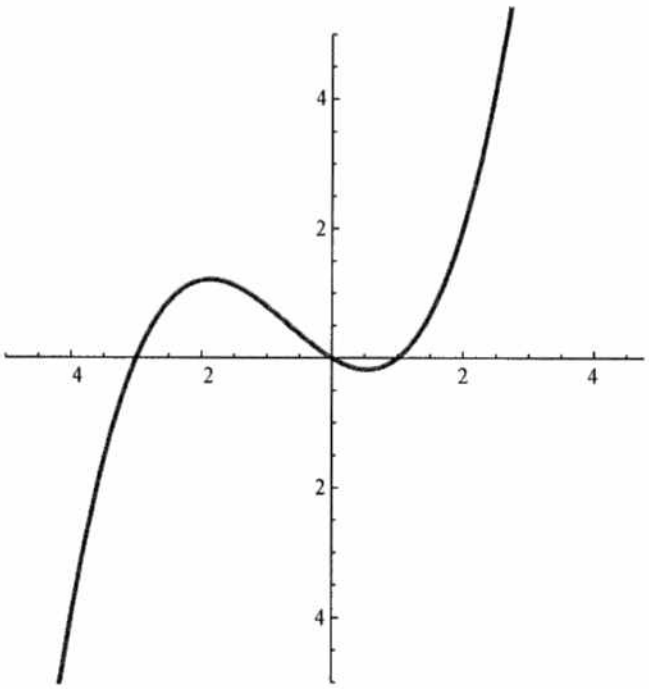
QUADRATIC

$y = f(x) = x^2$



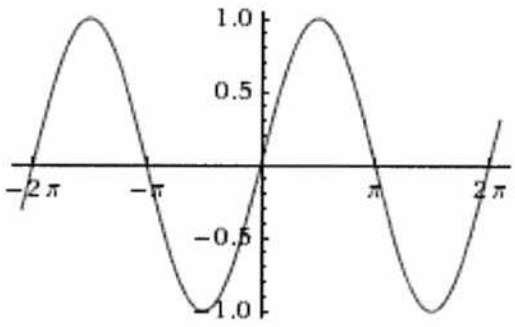
CUBIC

$y = f(x) = x^3$



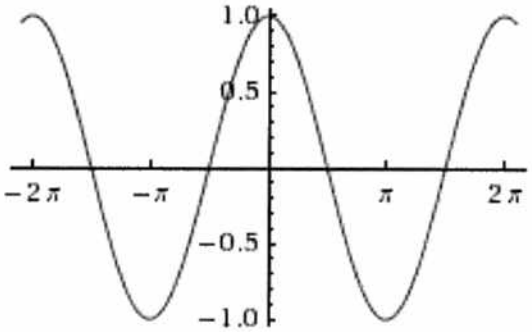
SINE

$y = f(x) = \sin(x)$



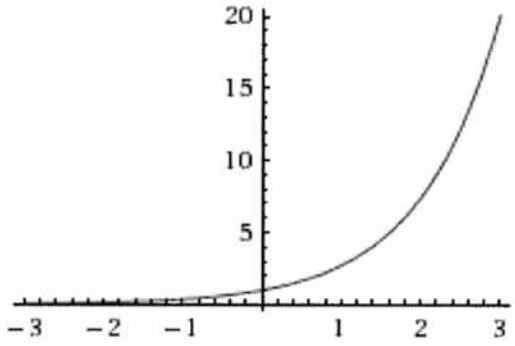
COSINE

$y = f(x) = \cos(x)$



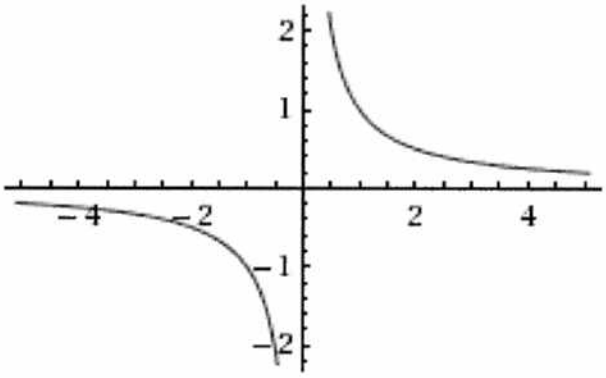
EXPONENTIAL

$y = f(x) = e^x$



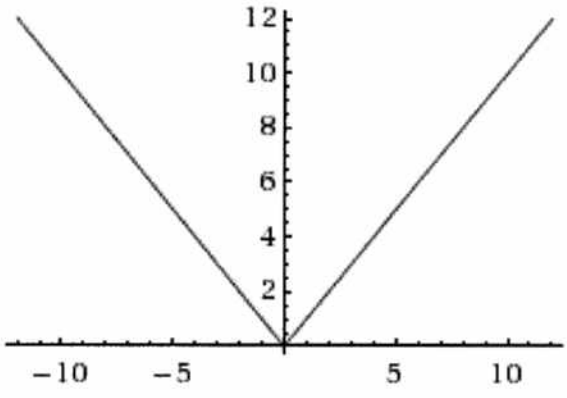
INVERSE

$y = f(x) = 1/x$



MODULUS

$y = f(x) = |x|$



1. INTRODUCTION

1.1 BACKGROUND

The problem concerns the Mathematics department at the [redacted] School, [redacted]. At present, the school has about 1,810 pupils, aged 11 – 18, making it one of the largest schools in [redacted]. At this moment in time the Mathematics department consists of 12 teachers who teach Key Stage 3, GCSE, GCSE Additional, A-level and Further Mathematics. My project would mainly relate to GCE Mathematics (specifically AS-level) students. In the academic year 2009/10 there are 127 AS-level Mathematics students. I myself have completed the A-level Mathematics course and am currently studying A-level Further Mathematics. Therefore, I have personal experience of the topic and understand various problems associated with learning it.

4	Graphs of functions: sketching curves defined by simple equations. Geometrical interpretation of algebraic solution of equations. Use of intersection points of graphs to solve equations. Knowledge of the effect of simple transformations on the graph of $y = f(x)$ as represented by $y = af(x)$, $y = f(x) + a$, $y = f(x + a)$, $y = f(ax)$.	Knowledge of function notation. Plotting graphs on graph paper will not be required.

Figure 1 [redacted] A-level Mathematics Specification

As you can see from the above excerpt and others (see Appendix section 1.3) the [redacted] A-level Mathematics course requires its candidates to understand Transformations of the graphs of functions, such as the ones shown in the "Explanation of Terminology" section. A Mathematics teacher, Mr V[redacted], has to teach this topic which is formally known as "The Transformations of the graphs of Functions." He, along with the rest of the Maths department, employ one of the following methods:

1. Write down prepared notes on a white board and ask students to copy them down. These notes contain still-images of the graphs of functions and their transformation, which the teacher sketches on the white board also. The teacher then explains the major concepts in the topic to the students.

2. Photocopy prepared notes and hand them out to each student. Still-images of graphs of functions and their transformations are also photocopied and distributed. After which, there is an explanation of the major concepts in the topic by the teacher.

1.2 PROBLEM DEFINITION

Teachers are not able to perform curve transformations in real time, meaning still images have to be drawn to try and explain various alterations to the initial function. This might make it difficult for the student to visualise certain transformations (such as the stretch ones shown in the terminology section) and how a transformation will affect a certain (x, y) co-ordinate on the initial function. This may lead to the student not fully understanding the topic, possibly affecting his/her performance in future Mathematics examinations

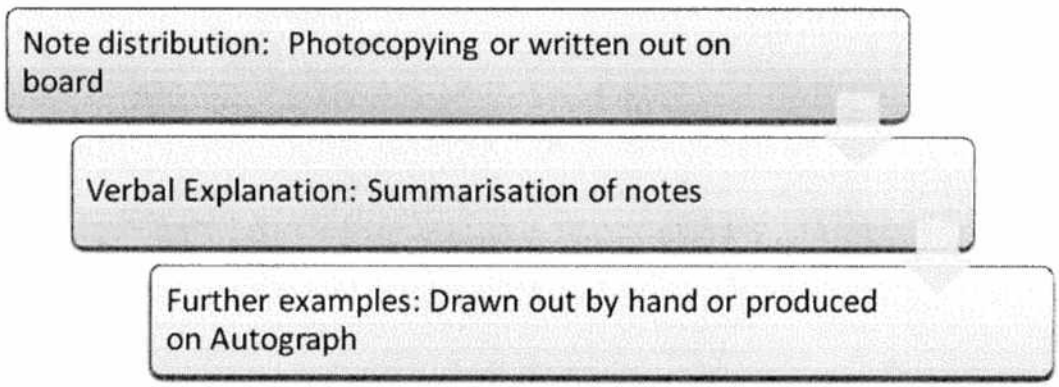
2. INVESTIGATION OF OLD SYSTEM

2.1 THE CURRENT SYSTEM

Since my program is essentially an aid to the teaching and understanding of the topic, the proposed system should then logically encompass the methods used by teachers used to actually explain the topic. Since my end-user preferred not to do an interview, I created an in-depth questionnaire based on interview questions that I normally would have asked.

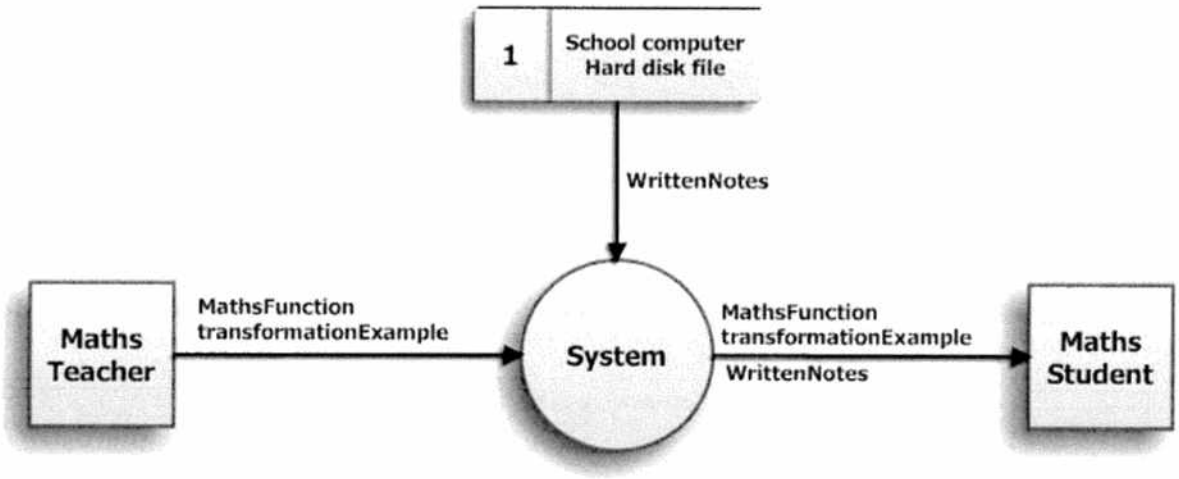
From this questionnaire (see Appendix section 1.1) and my own observations I gathered that most teachers have a 'stock' set of written notes (see Appendix section 1.2) for the transformations of functions and a few have their own hand-written notes. Those teachers with a 'stock' set of notes (used by the whole department) would photocopy them, according the number of pupils in the class, and then distribute them. On the other hand those using hand-written notes would usually write them on the board (though some photocopy) and ask students to write them down.

Regardless of the method of distribution, both sets of notes would contain the different types of function transformation, explanations of the effects of the transformations and graphical examples of each type. To further augment and better explain the notes, the teacher would also manually draw further examples on board or use Autograph software, which plots graphs, on a computer, that are then displayed on a projector screen. These further examples are still-images of functions that the teacher is able to recall from memory. The teacher then draws still images of the transformations for each type of function shown, explaining to the students what has happened to the original graph. Alternatively, these examples can be picked right out of the school Maths textbook - Introducing Pure Mathematics (Smedley & Wiseman, 2001) – and drawn using a boardmarker on a whiteboard or translated into Autograph, which draws it automatically. So to summarise:

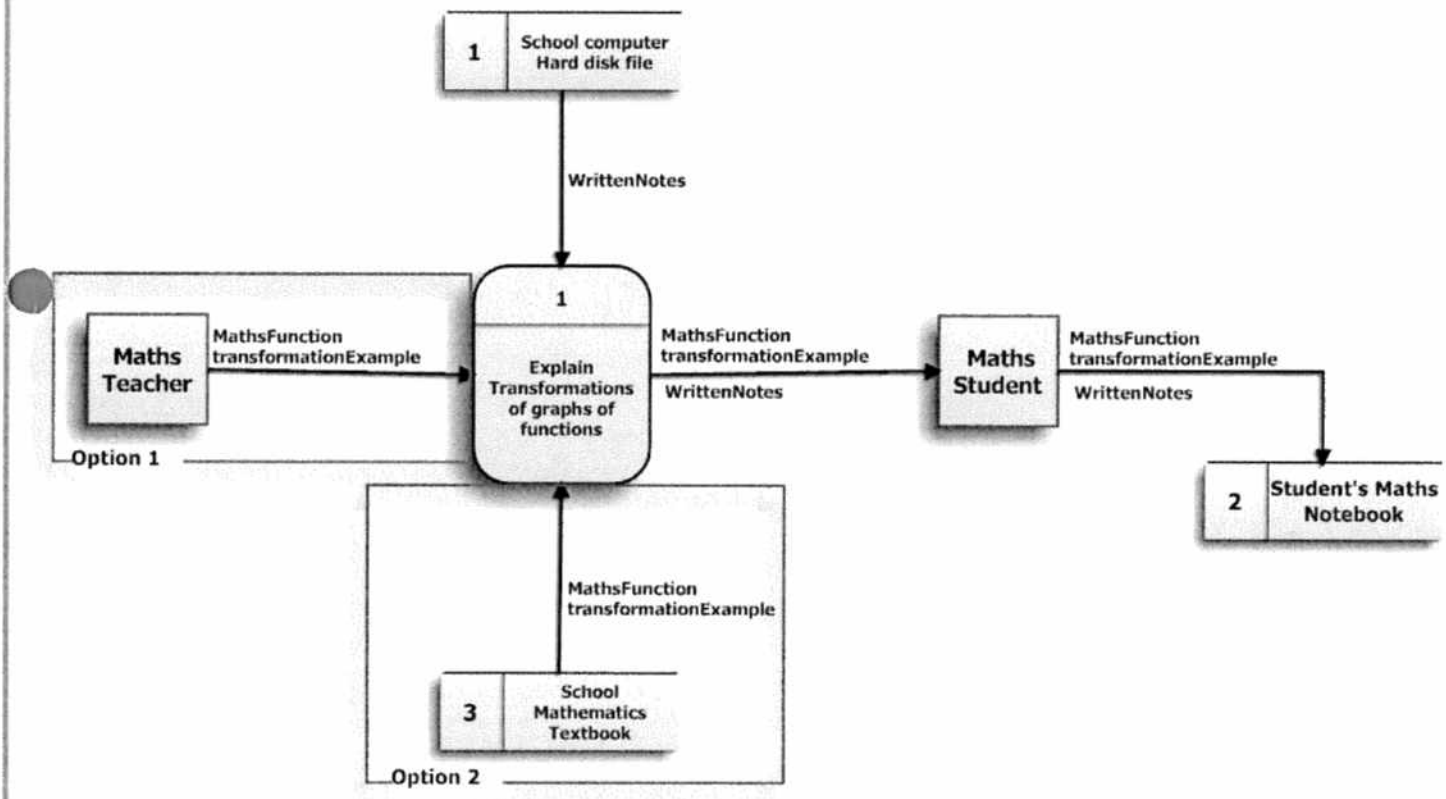


2.2 DATA FLOW DIAGRAM OF EXSISTING SYSTEM

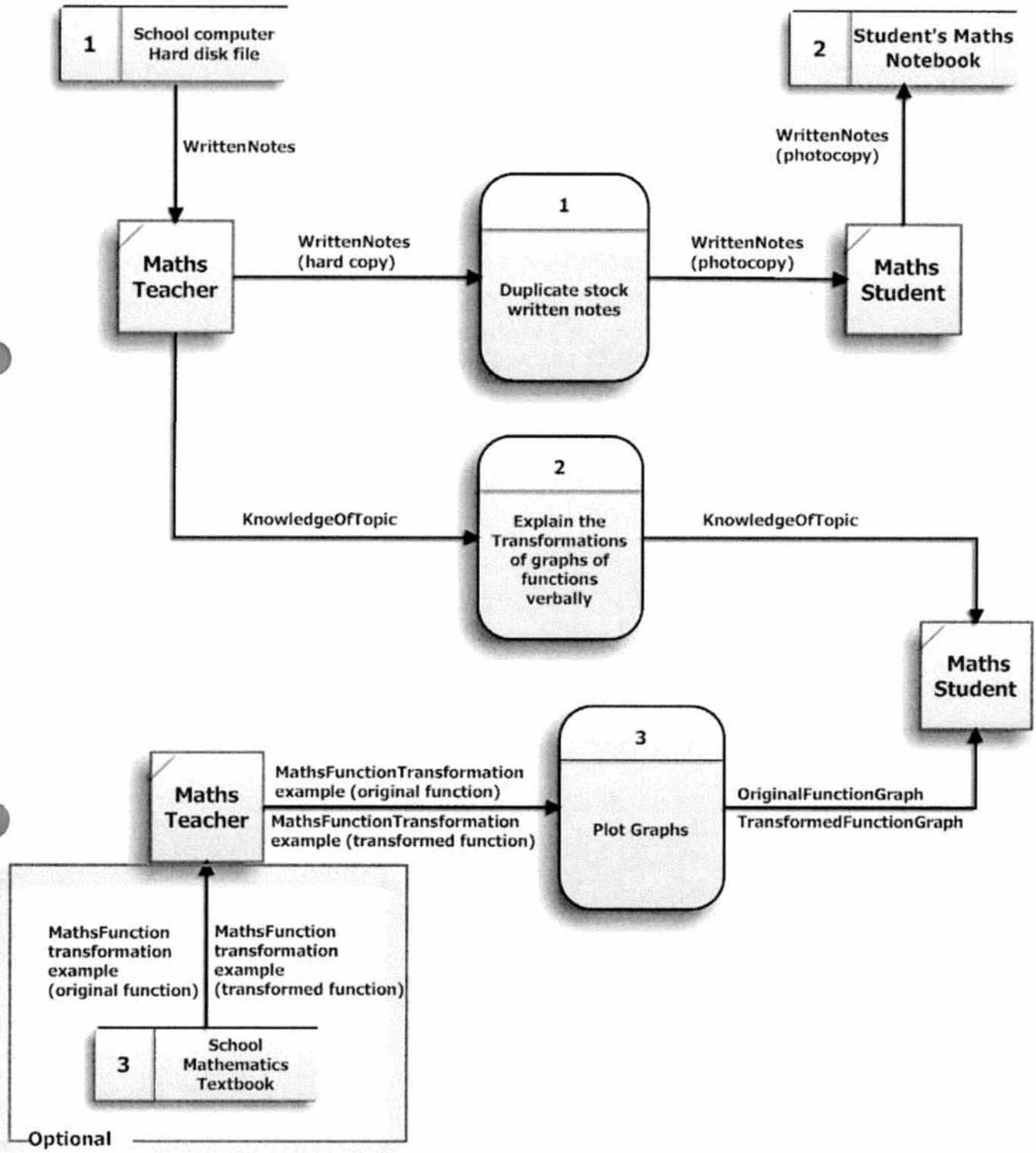
CONTEXT DIAGRAM



LEVEL 1



LEVEL 2 DFD


1
School computer Hard disk file
2
Student's Maths Notebook
WrittenNotes
WrittenNotes (photocopy)
Maths Teacher
WrittenNotes (hard copy)
1
Duplicate stock written notes
WrittenNotes (photocopy)
Maths Student
KnowledgeOfTopic
2
Explain the Transformations of graphs of functions verbally
KnowledgeOfTopic
Maths Student
Maths Teacher
MathsFunctionTransformation example (original function)
MathsFunctionTransformation example (transformed function)
3
Plot Graphs
OriginalFunctionGraph
TransformedFunctionGraph
MathsFunction transformation example (original function)
MathsFunction transformation example (transformed function)
3
School Mathematics Textbook
Optional

Findings from the questionnaire (see Appendix section 1.1 Enter Page No.)

- Referring to the answer in section 1- Q2, it is evident that the concepts underpinning, "The Transformations of the graphs of functions", aren't fully grasped and understood since:
  - The teacher, himself says that it is one of most difficult topics to understand
  - Frequent mistakes are made by students, due to their lack of understanding of what function transformation type corresponds to what graph.
- In section 1 - Q4, the still-image graphs drawn/shown by the teachers are proven to be ineffective. This is supported by the fact my end-user states that more interactive, dynamic examples are required; the reason being, that it makes it easier for students to visualise the concepts of the topic.

Findings through observation and own experience

- I find that the process of producing the further examples is slightly time consuming.
  - Drawing them out on the whiteboard requires the teacher to continually cross-reference a maths textbook or their own written notes (for accuracy of copying), and the whole process can take up valuable class time.
  - Continually photocopying the examples for each class taught the topic, can also take up a fair amount of time.
- In addition photocopying examples can also waste resources and lead to further unnecessary costs for the school.

3. INVESTIGATION FOR THE NEW SYSTEM

3.1 IDENTICATION OF THE PROSPECTIVE USERS

Mathematics department: A-level Mathematics teachers

End User: Mr W [redacted] (Mathematics Teacher)

3.2 IDENTIFICATION OF THE USER REQUIREMENTS AND ACCEPTABLE LIMITATIONS

Since the Mathematics department teachers are my only prospective users, only one set of user requirements are needed. They are as follows:

PROCESS REQUIREMENTS

The system must:

- Produce visual representations of graphs of functions.
- Produce real time transformations of the graph of a function including translations, stretching, reflection, inverse functions and combinations of these.
- Display transformations of the graphs of different functions encountered in A-level mathematics, not just one curve.
- Contain a basic explanation of the nature and magnitude of the transformation of the graph. For instance, the system should produce a sentence stating the graph has been translated by x-units parallel to a specific axis, where x is variable. Similarly, a sentence should be produced for different transformations.
- Produce the graph of the original function alongside its transformation for comparison.
- Have an option so that the user may print out the plots of the graphs drawn at that instant.
- Have an interface which is transparent and easy to follow. The graphical representation must encompass a majority of the program window to make the transformations easier to see and follow.

The user must:

- Be able to vary the magnitude and type of transformation.

- Be able to select a certain point on the graph of the transformed function and see its (x, y) co-ordinate. When such a point has been selected its previous position on the graph of the initial, untransformed function must also be highlighted so that the comparison is easier to see.

ACCEPTABLE LIMITATIONS

- The system can only deal with a finite range of x and y coordinates to plot the function.
- I will only be able to graph a small selection of the Functions that are encountered by students in A-level Mathematics, not all of them.
- The system can only vary the magnitude of the transformations to a finite amount.
- Since my project is essentially a learning aid and not a full substitute to the process of understanding the topic, I won't include the full level of detail required for a thorough explanation i.e. I won't use the written notes word for word.

3.3 DATA SOURCES, STORES AND DESTINATIONS FOR THE PROPOSED SYSTEM

STORES

- **A level Mathematics Textbook: Introducing Pure Mathematics** (Smedley & Wiseman, 2001) - This will come from my school's Mathematics department. I will choose the functions that are used in my program from this textbook. The functions will be entered into the system during the programming process, possibly as subroutines.

SOURCES

- **Maths Teacher -** They will select the type of function, the x co-ordinate on the plot and vary the magnitude of function transformations. At the outset of the program the type of function will most likely be selected from a title screen of fixed tabs, then if the user wishes to change his/her selection, the function can be chosen from a toolbar beside the plotted graph. The x co-ordinate and magnitudes will be varied using a slider or the user could directly input a specific value within an allowed range into a box.
    - **The functions** – e.g. mathematical ones such x^2 (quadratic), x^3 (cubic), e^x (exponential), etc. I will specify exactly all the functions I intend to use later

on in the project. I will select these from the A-level Mathematics textbook. Most likely, the user will be able to choose from a selection of many on screen using tabs or something similar.

- **The type of function transformation** – e.g. $f(x) + a$ (where $f(x)$ is the original function of x and a is a variable). This example and other types will be displayed as headings in a panel, each most likely having sliders and boxes underneath them. Thus, the type of function transformation isn't selected by the user; he/she only needs to vary the magnitude. Initially the magnitudes of all function types are set to zero.
- **The magnitude of function transformation** – e.g. for transformations like $f(x) + a$, a slider will be used to vary the value of the variable, a between a certain range. For those which don't have a magnitude to vary (e.g. $f(-x)$) a tick box will be used instead.
- ***x*-coordinate on plot**– Different values of x corresponding to a value of $f(x)$ on the plot will be selected using a slider. This is required so that a point (with x value selected by the user) on the original graph and its corresponding point on the transformed graph are highlighted. This makes the effect of the transformation on the (x, y) coordinates easier to see and allows mathematical comparison.

DESTINATION

- **Students** - Once the processes of the program are complete, the output will be displayed to the students (in actual fact the teacher as well). The output will include the:
  - **Plot of the graph of the original function** - This remains fixed throughout the transformations, so that comparisons can be made between the original and the transformation. It only changes when the user selects another type of mathematical function to be transformed
  - **Plot of the graph of the transformed function** – This changes according to the sliders or tick boxes which correspond to function transformation magnitudes and types.
  - ***y*-coordinate on plot** – A y co-ordinate on the original and transformed plots will be highlighted corresponding to the x co-ordinate input.
  - **Cartesian coordinates of highlighted point on graphs of the original and transformed function**– The points on the graphs of the original and

transformed graphs that are currently highlighted will have their Cartesian coordinates (i.e. x, y positions) output in a textbox.

- **Function transformation description** – A few sentences describing the type of function transformation(s) and the effect(s) on the graph of the initial, unaltered function.

3.4 DATA VOLUMES

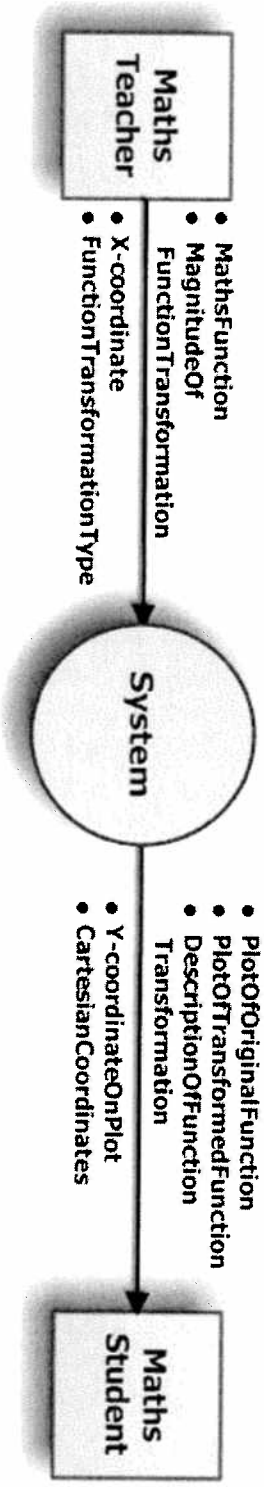
- **The functions** – 9 functions
- **The type of function transformation** – I will use 7 different types. These include: (see 'Explanation of terminology' section of analysis for further explanation)
  1. $f(x) + a$: A translation of $'a'$ units parallel to the y-axis (i.e. move the graph vertically up or down depending on the size and sign of $'a'$)
  2. $f(x + a)$: A translation of $'a'$ units parallel to the x-axis (i.e. move the graph horizontally left or right depending on the size and sign of $'a'$)
  3. $af(x)$: A stretch of the graph, parallel to the y-axis by a scale factor of $'a'$
  4. $f(ax)$: A stretch of the graph, parallel to the x-axis by a scale factor of $\frac{1}{a}$
  5. $-f(x)$: A reflection of the graph in the x-axis (horizontal axis)
  6. $f(-x)$: A reflection of the graph in the y-axis (vertical axis)
  7. $\frac{1}{f(x)}$: The graph of the inverse of $f(x)$

- **The magnitude of function transformation** – The magnitude will vary between -10 and 10, with a transformed point being plotted for every 1 unit interval.
- **x co-ordinate on plot** – The value of the x co-ordinate depends on the range of the x-axis of the plot. The x-axis will have a range of -10 to 10.
- **y co-ordinate on plot** - The value of the y co-ordinate depends on the range of the y-axis of the plot. The y-axis have a range of -10 to 10.

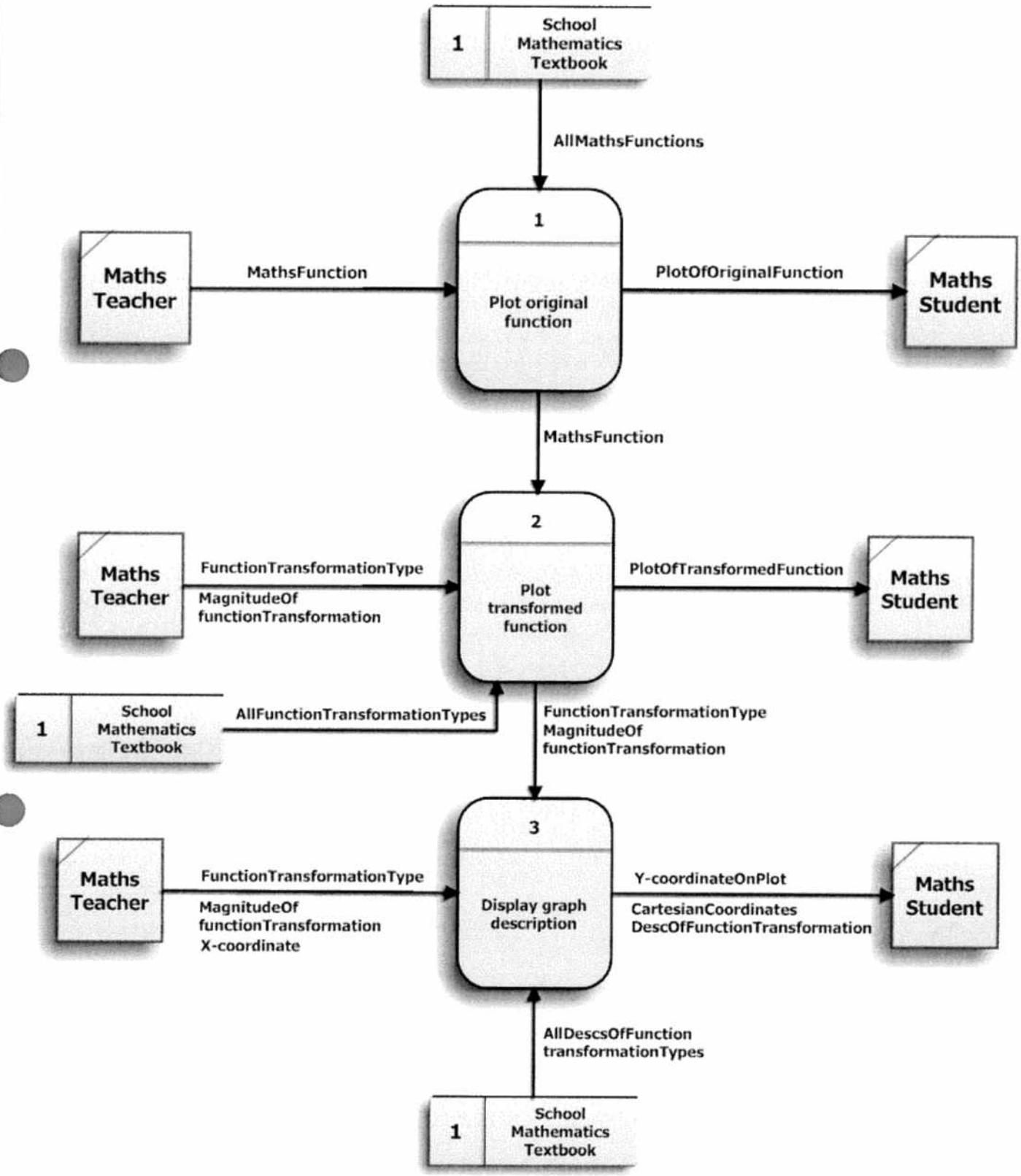
- **Cartesian coordinates** – The x-values of the Cartesian coordinates depend on the range of the x-axis, so it will range from -10 to 10. However the y-values of the Cartesian coordinates are dependant of the effect the functions have on the x-values, making it difficult to calculate the exact range.

3.5 DATA FLOW DIAGRAM OF PROPOSED NEW SYSTEM

CONTEXT DIAGRAM



LEVEL 1 DFD


1
School Mathematics Textbook
AllMathsFunctions
1
Plot original function
Maths Teacher
MathsFunction
PlotOfOriginalFunction
Maths Student
MathsFunction
2
Plot transformed function
Maths Teacher
FunctionTransformationType
MagnitudeOf functionTransformation
PlotOfTransformedFunction
Maths Student
1
School Mathematics Textbook
AllFunctionTransformationTypes
FunctionTransformationType
MagnitudeOf functionTransformation
3
Display graph description
Maths Teacher
FunctionTransformationType
MagnitudeOf functionTransformation
X-coordinate
Y-coordinateOnPlot
CartesianCoordinates
DescOfFunctionTransformation
Maths Student
AllDescsOfFunction transformationTypes
1
School Mathematics Textbook

3.6 DESCRIPTION OF PROCESSES AND DATA FLOWS

Looking at the level 1 data flow diagram:

PROCESSES

1. Plot original function

Here the teacher chooses a mathematical function to be plotted (and eventually transformed). The program, using the relevant sub-routines, plots the corresponding graph of the function which is then displayed on the graphical user interface (GUI).

2. Plot transformed function

Depending on what type and magnitude of function transformation, the program would carry out and plot (on the same axis as the original graph of the function) the transformed function.

3. Display graph description

This process generates and displays text regarding the type/s transformation applied, their effect geometrically (e.g. "the graph has be translated three units to the right") and the effect on the coordinates (e.g. "the y-coordinates have been halved to incur this transformation" or similar words to that effect). In addition this process

DATA FLOW

Process 1

- The process has the data of all the possible functions input from the Mathematics Textbook. (This only ever happens once)
- The Teacher selects the required function, which is input to the process
- The process then converts the above input into a plot of the graph of the function, which is output on screen

Process 2

- All the needed function transformation types are taken from the school's mathematics textbook. (This only ever happens once)
- The teacher then alters the magnitude using a slider of the corresponding function transformation, which is read by the process.
- The process then outputs a new plot of the function (selected in process 1) with all the required transformations to the GUI, which the students can view through a class projector

Process 3

- Descriptions of all the function transformation types are initially input during the programming stage.
- The function transformation types and magnitudes which have already been input earlier are taken by the process. However the teacher is required to specifically input the x-coordinate of point to be highlighted on both plots.
- The process then outputs the y-coordinate of the point to be highlighted and a on-screen description of the function transformation/s.

3.7 ANALYSIS DATA DICTIONARY

Name	Description	Data Type	Size/range
PlotOfOriginalFunction: ***x*-axis range**	In order for the graph of the function selected by the Teacher to be plotted, the x-axis range will have to be specified by the programmer during the implementation.	Real	-10 to 10
PlotOfOriginalFunction: ***y*-axis range**	In order for the graph of the function selected by the Teacher to be plotted, the y-axis range will have to be specified by the programmer during the implementation.	Real	-10 to 10
MagnitudeOfFunction Transfomormation	This corresponds to the amount the slider has been moved	Real	-10 to 10
***x*-coordinate**	The x value chosen by the user, corresponding to the points to be highlighted on the plots.	Real	-10 to 10
***y*-coordinateOnPlot**	The y-coordinates of the points corresponding to the user's selection of x value	Real	-10 to 10
CartesianCoordinates	Text displaying the (x, y) coordinates of the point highlighted	Real	Unsure
DescOfFunction Transformation	Text explaining the function transformation incurred on the original function	String	2 to 3 short lines of text

4. OBJECTIVES FOR THE PROPOSED SYSTEM

Taking into account user requirements, old and new system analysis and my questionnaire with the end-user, I have produced a set of objectives. Before I state my specific objectives, I feel it is necessary to state my overall objective:

To create a program which will act as a teaching/learning aid for the topic of "Transformations of the graphs of functions", by plotting and automating graphical interpretations of specific functions.

4.1 INPUT, PROCESSING AND OUTPUT REQUIREMENTS

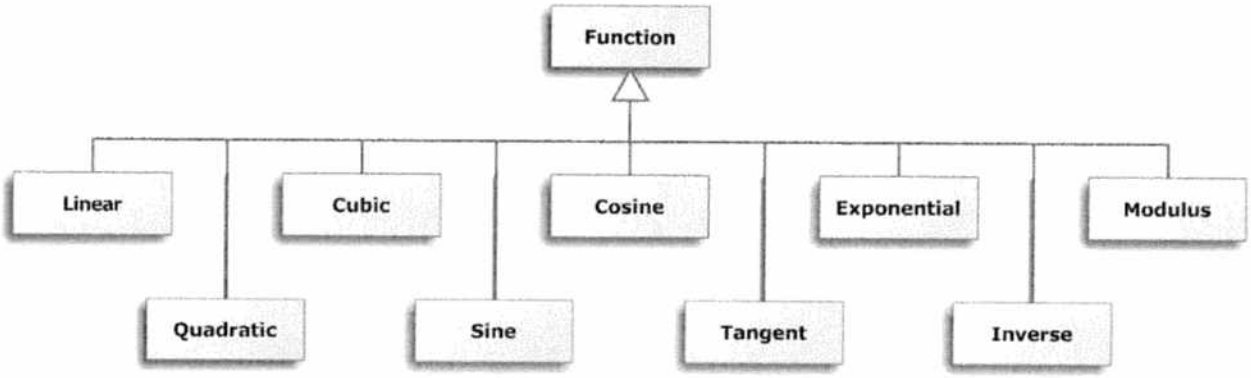
1. The system must be able to plot and output the original and transformed graphs on the same axes of 9 different mathematical functions. Following on, the respective x and y axes must also have a range of at least -10 to 10 each.
2. The system must be able to produce seven different types of function transformations: $[f(x) \pm a]$, $[f(x \pm a)]$, $[af(x)]$,$[f(ax)]$, $[-f(x)]$, $[f(-x)]$ and $[\frac{1}{f(x)}]$.
3. For transformations that can have varying magnitudes such as $[f(x \pm a)]$, the range of the variable $'a'$ must range from -10 to 10.
4. The system must be able to produce text describing the effect and magnitude (if applicable) of the type of transformation/s currently applied on the selected function. The text must also change the value of the magnitude it states as the user varies it accordingly.
5. The system must be able to highlight a point on the plots of both the original and transformed graphs corresponding to the user's selection and also output the Cartesian coordinates of that point.
6. The system must be able allow the user to print a screenshot of the graph area

4.2 PERFORMANCE REQUIREMENTS

1. The plot of the transformed graph must update smoothly and without flicker as the magnitude of the transformation is varied.
2. For those transformations without magnitude the plot should update in less than two seconds when the transformation is applied (e.g. after a tick-box corresponding to the transformation is clicked).

5. OBJECT ANALYSIS DIAGRAM

INHERITANCE DIAGRAM



A mathematical function can be seen as an object, so it would make sense to create a class for them. On the other hand, plotting the graphs of functions and transformations on them can be seen as Methods within the class definition.

6. REALISTIC APPRAISAL OF THE FEASIBILITY OF POTENTIAL SOLUTIONS

MANUAL (PRE-EXISTING) SOLUTION

This is the current system. As I have documented already, it involves the teacher drawing still-images of the graphs of functions on the blackboard. Then, for each transformation type, the graphs of the transformed functions are drawn. Alternatively, the teacher could also draw the graphs using Autograph software (which I will document later). I have already explained the problems with the system earlier on in my Analysis but I will continue to give a brief summary of the advantages/disadvantages:

Advantages:

- Minimal IT knowledge required in comparison to rest of the potential solutions (current Maths staff can use Autograph at a very basic level)

Disadvantages: (see analysis section 2.3- Discussion of problems with current system.)

- It is quite time consuming to manually draw out the graphs
- The students don't seem to grasp easily the concept through this method, as frequent mistakes are made by students, due to their lack of understanding of what function transformation type corresponds to what graph.
- Doesn't produce interactive and dynamic examples

OFF-THE-SHELF SOFTWARE SOLUTIONS

AUTOGRAPH

Autograph is a piece of dynamic graphing software used for teaching mathematics at secondary level. The current version is 3.2 and it is bought on license to independent users or schools. It can plot graphs in all three dimensions and operates in three main modes corresponding to each dimension. Its main uses are in Statistics, Probability, Graphing and Transformations, where it can be used to transform the graphs of mathematical functions.

Homepage: http://www.autograph-math.com/index.shtml

Advantages:

- It is able to plot to graphs of any mathematical function input into the system
- Through the use of a feature called the "constant controller", the graphs of functions can be transformed quite smoothly
- Graphs can be plotted and transformed in three dimensions

Disadvantages:

- There are no set buttons or sliders assigned to mathematical functions. For instance, just to carry out one transformation you have to apply a series of steps (which you need to understand), in comparison to just moving a slider or ticking a check box.
- It can't apply two or more transformations to a function
- A license can cost up to £650 for a school/college. Even though the school currently possesses the software, there is a disadvantage in the fact that many of Autograph's features aren't fully utilised.
- Some of the program's features are quite advanced and require a substantial amount of IT knowledge. This could pose a problem considering the limited IT skills of the potential users (Maths Department).

OMNIGRAPH FOR WINDOWS

Omnigraph is a graph processor, now at Version 2, published by SPA. Besides plotting graphs, analysing gradients and solving differential equations, it can also draw polygons and reflect, rotate, enlarge or translate them. Omnigraph also allows the transformations of graphs of functions using a tool called a "dynamic constants editor".

Distributor homepage: http://www.virtualimage.co.uk/html/omnigraph.html

Advantages:

- It loads and runs very quickly.

- It is very easy to use; you just type the function into a textbox and it plotted almost simultaneously.
- It can plot and transform almost any type of mathematical function

Disadvantages:

- Like Autograph, there are no dedicated buttons or sliders assigned to "Function transformations". These dedicated buttons are important because they decrease the amount of steps the user has to go through, to transform graphs.
- It can't apply more than one function transformation.

BESPOKE SOFTWARE SOLUTIONS

PROGRAMMED SOLUTION

This is a program which I will create myself using my selected language. I will develop the software by following the Waterfall development model.

Advantages:

- Will tailor to the end-user's requirements closer than the previous solutions
- It should fulfil objectives created by the programmer and decided on by the end user (see 'Objectives' section)
- Should be completely dedicated to the topic of the "Transformations of graphs of Functions"
- Can apply more than one function transformation (i.e. compound function transformations)

Disadvantages:

- Developed by a beginner programmer
- No software updates

PROGRAMMING LANGUAGES

VISUAL BASIC

Since my project involves a fair amount of graphical/visual work, I feel Visual Basic is a language worth considering.

Advantages:

- Visual Basic was designed to be easily learned and used by beginner programmers
- It has a large variety of graphical development features
- Copious amount of coding isn't required since the language mainly involves arranging pre-existing components or controls on a form.

Disadvantages:

- Poor support for object-orientated programming. However its predecessor, Visual Basic .NET, provides extensive support.
- Microsoft has ended its support for the language (except for Visual Basic .NET) and its development environment.
- The AQA Computing article, "Why chose Pascal for GCE computing?" states many disadvantages in the, "Visual Basic versus Pascal/Delphi", section. (webpage: http://store.aqa.org.uk/qual/gce/pdf/AQA-2510-W-TRB-OGWCP.PDF).
- I would have to learn the language from scratch.
- I feel Visual Basic wouldn't give me the complexity required for my project

PASCAL/DELPHI

Pascal is an influential imperative and procedural programming language with an object-orientated equivalent called Delphi. It is a language I am considering using because of my past experience of it in AS Computing and the fact that it is recommended by an article on AQA (webpage: http://store.aqa.org.uk/qual/gce/pdf/AQA-2510-W-TRB-OGWCP.PDF).

N.B. I will be quoting some of my advantages from the aforementioned article:

Advantages:

- "Pascal is a very easy programming language to understand"
- Pascal is an object-orientated programming language. This is useful for my project since I plan to use objects, e.g. a 'Function' can be treated as an object.
- "Many Computer Science/Computing departments have now switched to mini-languages based on Pascal-like languages"
- "Lazarus Pascal and Delphi are recommended for use in schools in mainland Europe"
- I have had previous experience in Pascal/Delphi

Disadvantages:

- Limited documentation and support for Lazarus

7. JUSTIFICATION OF CHOSEN SOLUTION

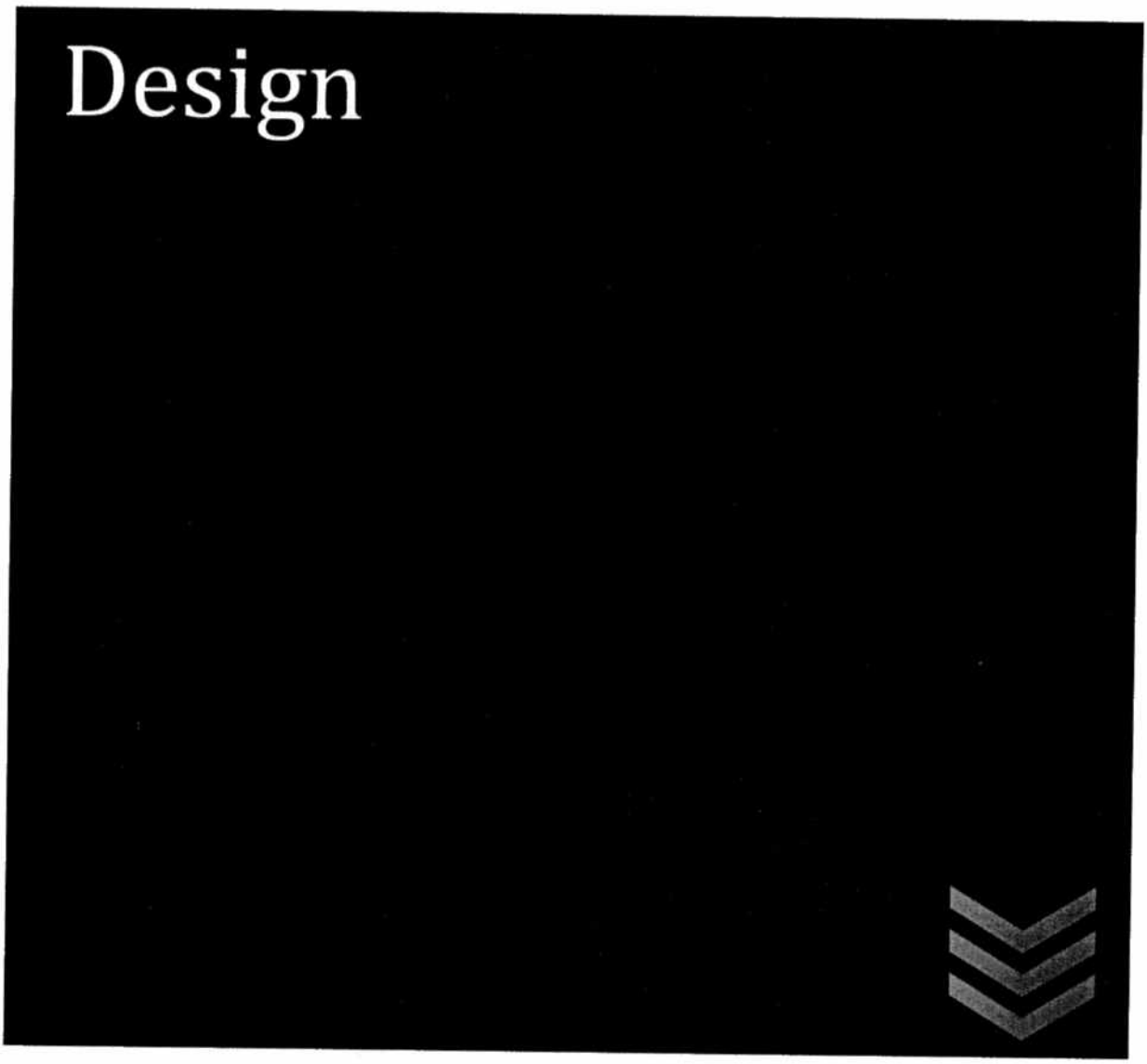
After carrying out observation of teaching methods, analysing the current/proposed system and consulting with my end-user, I have selected a solution. I have decided to create a computer program in the language, Pascal.

I have rejected the other currently available software solutions because I feel they don't fully meet the requirements of the end-user (as specified above in section 6). For instance, the fact that Autograph is an off-the-shelf software solution means it doesn't specifically concentrate on acting as a learning aid on the topic of the "Transformations of the graphs of Functions". This means the user has to navigate through the program interface and carry out more than one process to carry out a function transformation. The same would also apply to Omnigraph. In addition, both programs are unable to apply more than one function transformation simultaneously since they haven't been specifically designed to perform function transformations.

I have decided that the most suitable solution would be a programmed system because it allows me to take into account the user's requirements and can be made to comply with my chosen objectives. Also a programmed solution, unlike the others, will specifically cater to "The graphs of the transformations of functions", rather than treat it as an extra feature of the software. My chosen language is Pascal/Delphi because I have had some experience in programming with it before. It also allows me, through the use of a package called "PlotPanel", to plot the graphs of mathematical functions and I can use Pascal to transform the graphs, code the Functions and create the GUI. In addition, there are numerous reasons in the AQA article (mentioned above) why I should choose Pascal for my GCE Computing project.

To help me design and create my program, I will be using one piece of software. To edit and compile the program I will use the Open Source Lazarus IDE (Integrated Development Environment). As I said earlier, I am also going to use a Delphi component called "PlotPanel" to plot the basic shapes of the graphs that I will specify.

Design



Specification Excerpt:

Specify and document:

- The method of solving the problem including, where appropriate, evaluation of alternative proposals
- The functions of constituent parts of the system
- The inter-relationships between the various parts of the system
- The selection of an appropriate hardware and software configuration
- The algorithms, data types, data structures and any other requirements of the solution
- The effectiveness of the proposed solution in meeting the requirements of the problem.

CONTENTS

1. Overall System design 4

Overview 4

2. Description of modular structure of system 4

Hierarcy diagrams 4

system flowcharts 7

3. Design Data dictionary 11

4. Volumetrics 13

Calculation of disk space required 13

identification of storage media 13

Hardware specification 13

5. Identification of suitable algoritms 14

Plotpanel 15

Set Xy Axes limits 18

set xy axes intervals and layers 18

plot xy axes 19

plot function 20

get function transformation magnitudes 21

apply transformation 22

highlight points 24

Calculate transformed graph info 25

Show Graph info 26

calc and show cart coord 26

options box 27

reset graph 28

print graph 28

6. Class Diagrams, object behaviours and methods 29

class diagrams 29

Class definitions (These may change during implementation stage) 32

7. user inteface design 34

UI Sample of planned Input with Output Designs 34

UI sample of planned data capture and entry designs 37

User interface design rationale 39

8. Security and integrity of data 40

9. System security 40

10. Test Strategy 41

1. OVERALL SYSTEM DESIGN

OVERVIEW

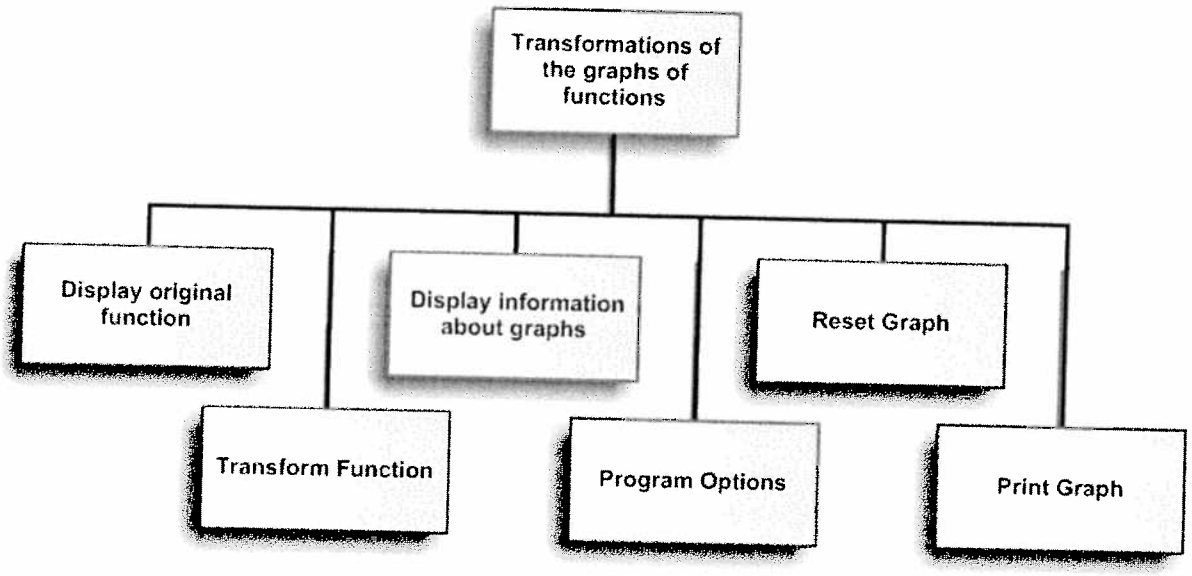
I am going to be designing an application which displays the graphs of functions and transforms them according to the user's input. Delphi (Object Pascal) will be used to code the program and it will be used by the school's Mathematics Department

The system has only one set of users - the Mathematics teachers. They will be able to select a mathematical function from a list and its graph will be displayed on screen. They can then chose a transformation type and adjust its magnitude (if it has one). As the magnitude is varied, a plot of selected function with the appropriate transformation is output on the same axis as the original graph. The user may then further transform the graph by varying the magnitudes of different transformation types. They also have the power to vary the x-coordinate of a highlighted point on both of the graphs. They should also have the option of resetting the graph back to its untransformed version, printing the graph and varying the different settings of the graph display (e.g. turning axis labels on or off).

2. DESCRIPTION OF MODULAR STRUCTURE OF SYSTEM

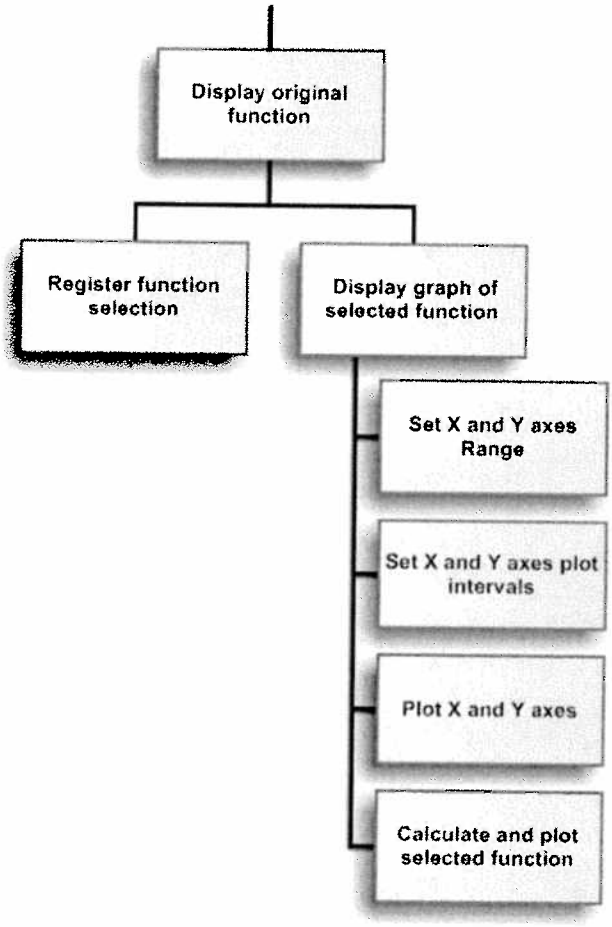
HIERARCY DIAGRAMS

FIRST TIER

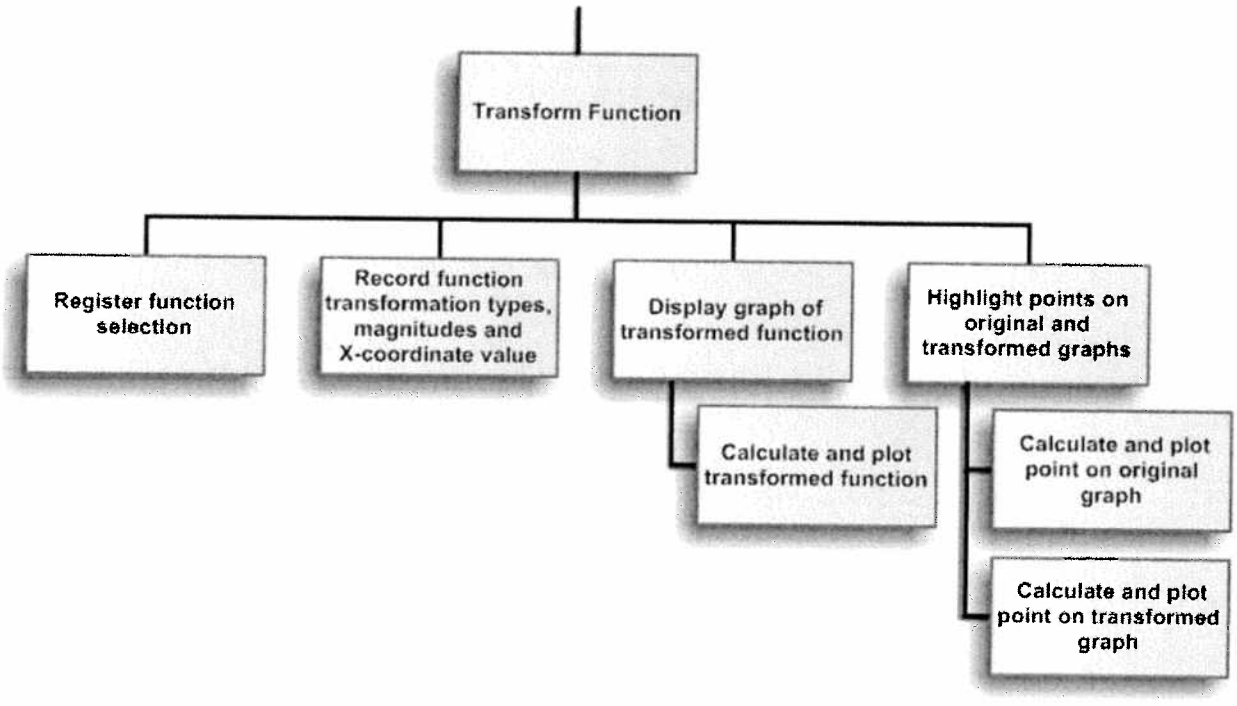


SECOND TIER

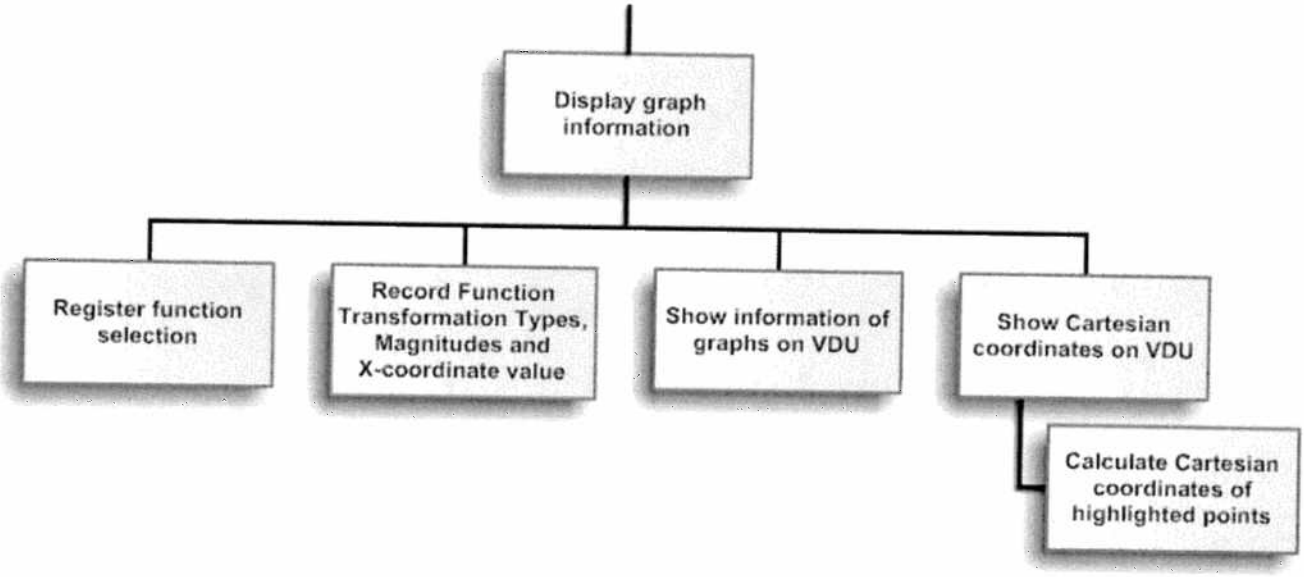
DISPLAY ORIGINAL FUNCTION



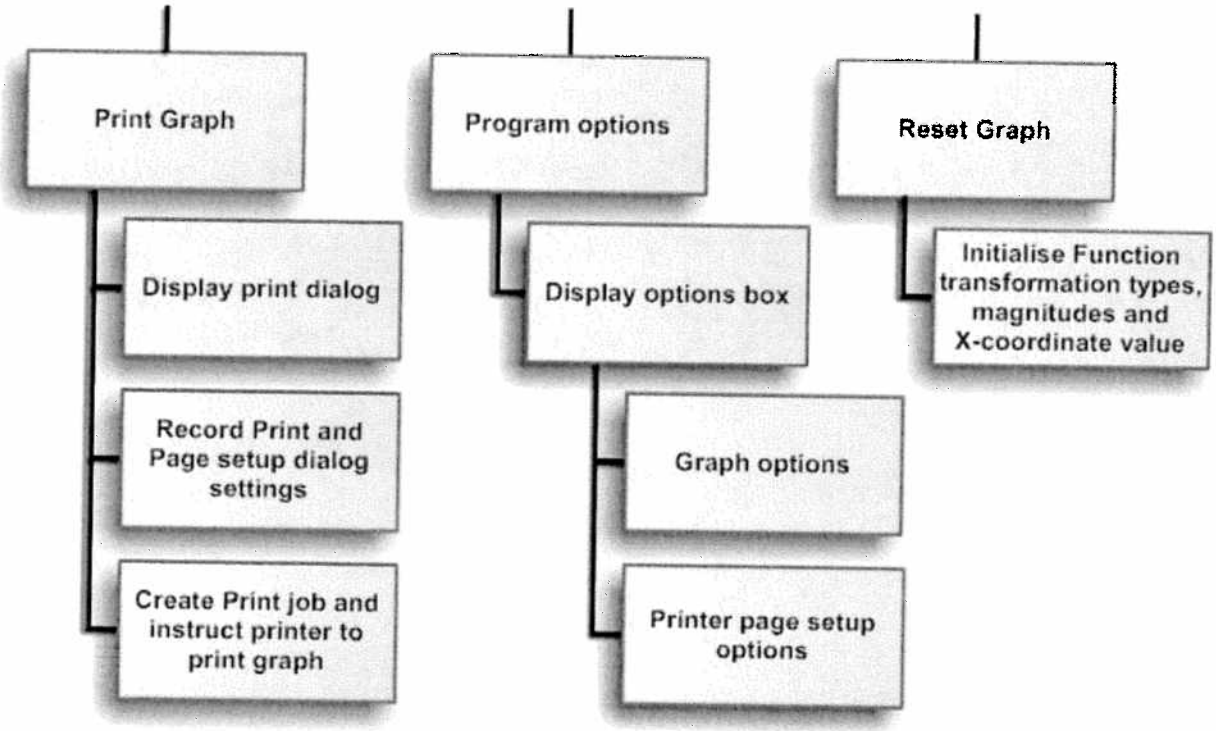
TRANSFORM FUNCTION



DISPLAY GRAPH INFORMATION



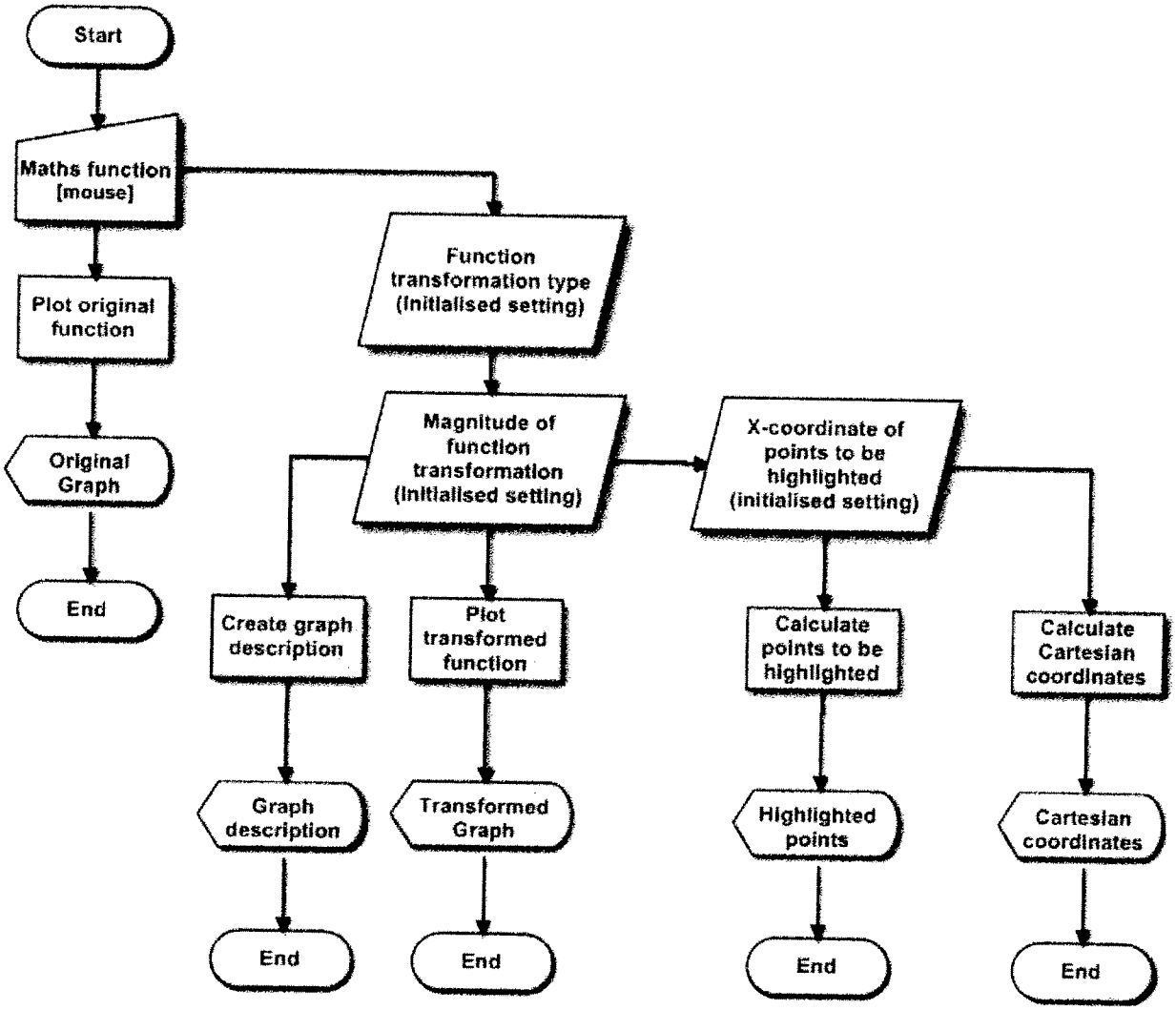
PRINT AND RESET GRAPH, AND PROGRAM OPTIONS



SYSTEM FLOWCHARTS

These are used to map the different pathways the user can take through the system. I.e. the different processes they can carry out

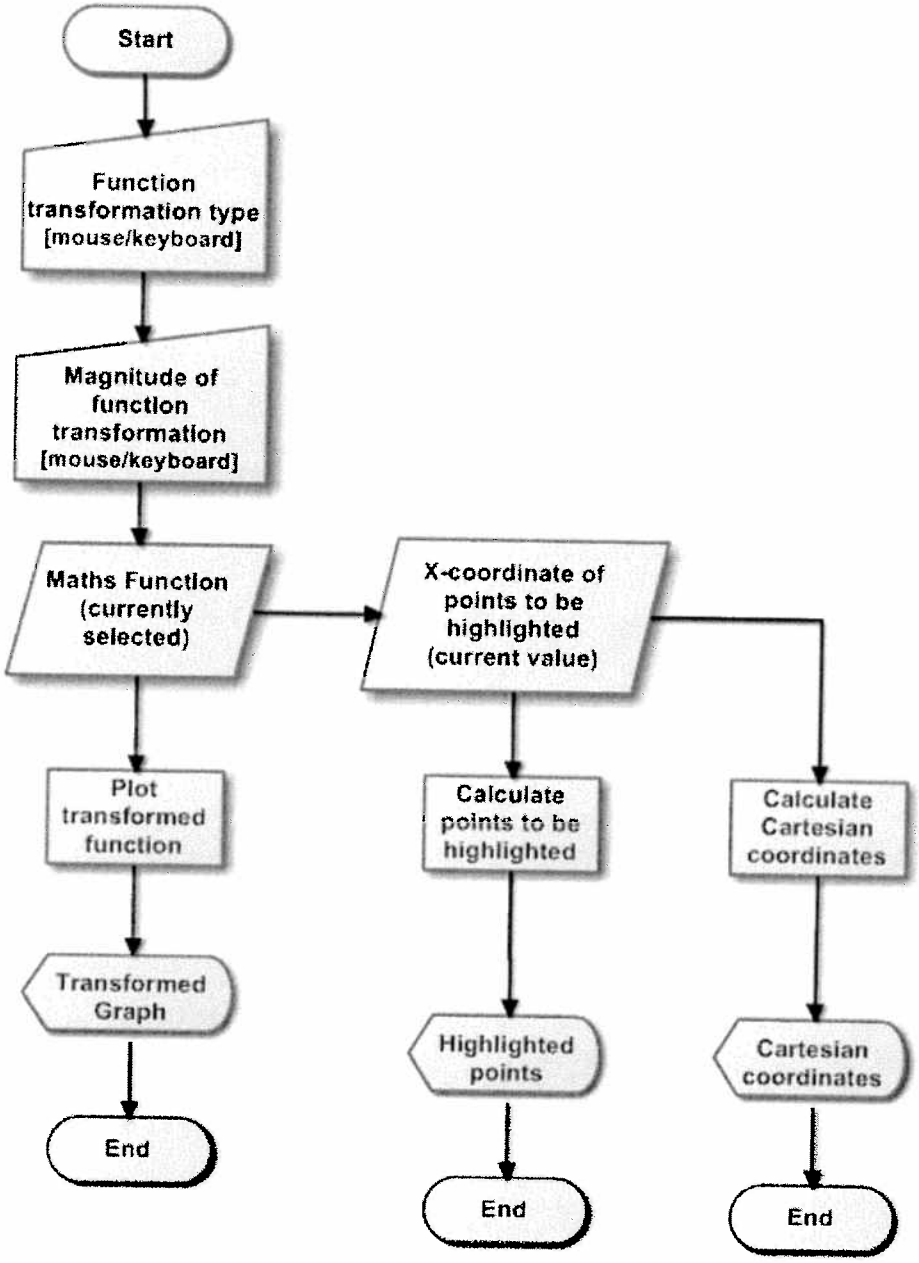
SELECTION OF MATHEMATICAL FUNCTION



This flowchart is used to signify the point right after the program is started-up, when the user selects a Mathematical function. I have used the term 'initialised setting' to show that the program is actually reading in the initial value that has been programmed into the system, not a value chosen by the user. This is so that a 'stock' transformed graph is displayed to the user so that he/she knows which graph is going to be transformed.

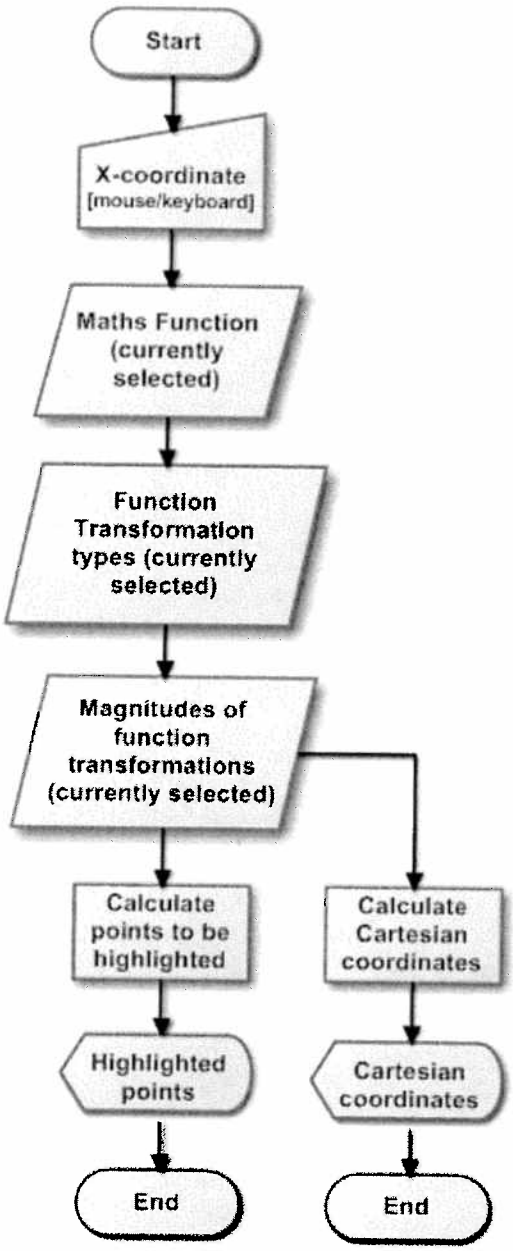
These processes, including the plot of the original graph, happen almost simultaneously therefore the user will see the combined output of each of the processes on the VDU.

TRANSFORMATION OF GRAPH



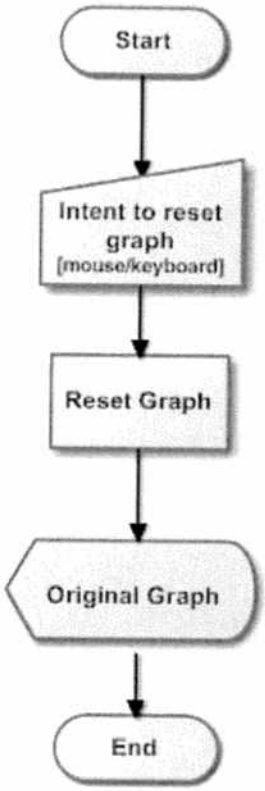
This happens after the user has selected a function. The user can either chose a transformation which has no magnitude or can chose to vary the magnitudes from a pre-displayed list of transformations on screen. By this I mean that the transformations with a magnitude don't have to be selected and then have their magnitudes varied, instead the user directly varies the magnitude of the transformation he/she requires. However the user has to select the ones without magnitudes as there is no other way to indicate that these are the transformations that the user requires. Once again this all happens nearly simultaneously.

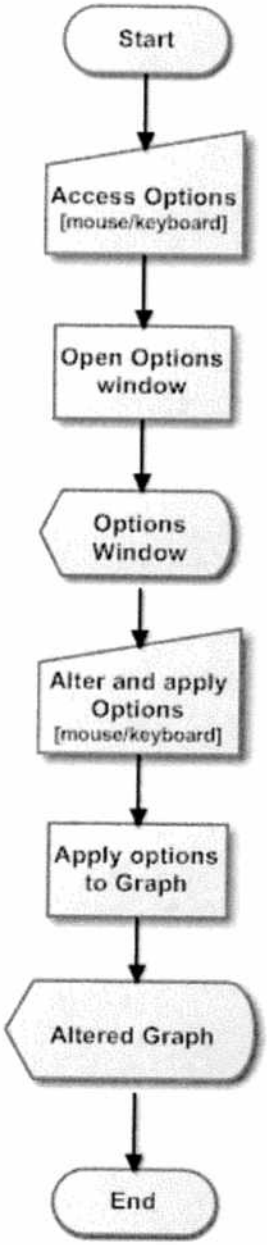
SELECTION OF X-COORDINATE

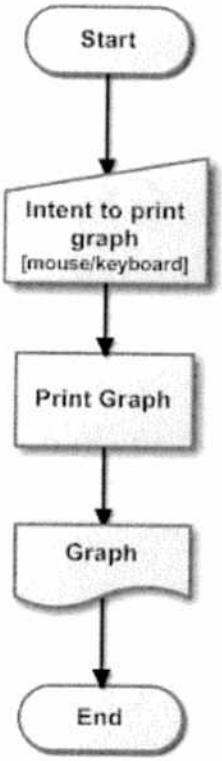


This flowchart demonstrates how the program highlights the points (with x-values selected by user) on the original and transformed graphs. The points that are highlighted are of the same x-value and are used to show how a transformation has affected a point on the original graph.

As you can see the program reads-in the 'function currently selected', 'the transformation types' and the 'magnitudes of the function transformations'. Only after all of these are read-in can the program start calculating the positions of the highlighted points and the y-values of the Cartesian coordinates, and then display them.







These flowcharts demonstrate optional processes which increase the functionality of the program and make it easier to use. For instance, the reset button initialises the graph back to its original form so a new set of transformations can be performed, and the options box allows for further customisation of the program.

3. DESIGN DATA DICTIONARY

Object Field Name	Data Type	Max Length	Validation required	Description
Xmin	Integer	4	None (Since input during coding process)	The minimum value the x-axis can take for the graph
Xmax	Integer	4	None (Since input during coding process)	The maximum value the x-axis can take for the graph
Ymin	Integer	3	None (Since input during coding process)	The minimum value the Y-axis can take for the graph
Ymax	Integer	3	None (Since input during coding process)	The maximum value the x-axis can take for the graph
YValueue	Extended	6	None (Since input during coding process)	Variable that is used to hold the result of a calculation that is used to find out the y coordinate of a function at a certain x coordinate
YTranslation (a)*	Integer	3	Data type check (Component automatically validates)	The magnitude of the function transformation $f(x) + a$
XTranslation (b)*	Integer	3	Data type check (Component automatically validates)	The magnitude of the function transformation $f(x + a)$
YStretch (c)*	Integer	3	Data type check (Component automatically validates)	The magnitude of the function transformation $af(x)$
XStretch (d)*	Integer	3	Data type check (Component automatically validates)	The magnitude of the function transformation $f(ax)$
XCoordinate(e)*	Integer	2	Data type check (Component automatically validates)	The x value chosen by the user, corresponding to the points to be highlighted on the plots.
ReflectX(f)**	Integer	2	None (Since input during coding process)	Used to signify whether tick box corresponding to transformation, $-f(x)$, has been clicked

ReflectY (g)**	Integer	2	None (Since input during coding process)	Used to signify whether tick box corresponding to transformation, $f(-x)$, has been clicked
InvX	Boolean	N/A	None(Since corresponds to tick box states)	Used to signify whether tick box corresponding to transformation, $\frac{1}{f(x)}$, has been clicked
XRed	String	2	None (Since output from program)	The x value of the Cartesian coordinate corresponding to point highlighted on the original graph
YRed	String	4	None (Since output from program)	The y value of the Cartesian coordinate corresponding to point highlighted on the original graph
XBlue	String	2	None (Since output from program)	The x value of the Cartesian coordinate corresponding to point highlighted on the transformed graph
YBlue	String	6	None (Since output from program)	The y value of the Cartesian coordinate corresponding to point highlighted on the transformed graph
Sa	String	3	None (Since output from program)	The string version of field '**a**' which will be used for the dynamic text information about the graphs
Sb	String	6	None (Since output from program)	The string version of field '**b**' which will be used for the dynamic text information about the graphs
Sc	String	3	None (Since output from program)	The string version of field '**c**' which will be used for the text information dynamic about the graphs
Sd	String	10	None (Since output from program)	The string version of field '**d**' which will be used for the dynamic text information about the graphs
TextInfo	String	20	None (Since output from program)	String combining **'Sa', 'Sb', 'Sc', 'Sd'** and other strings to produce dynamic text information about graphs

*I will use a single character variable identifier for this field because it makes it a lot easier to perform inevitable future calculations with it.

** I have used an integer data type for this field even though the function transformation it represents has no magnitude. This is because the effect of this certain function transformation is the same as multiplying part of or the whole function (currently being transformed) by -1. Thus when the tick box corresponding to this function transformation is checked I will make this field equal to -1 and when unchecked I will make it equal to 1.

4. VOLUMETRICS

CALCULATION OF DISK SPACE REQUIRED

Since my project is a non-database project I have no records or tables to store, therefore I don't have the same calculations to do for storage space requirements. However, my stand-alone application will require storage space in terms of the coding with the different sub-routines, forms, variables and class definitions. The data I mentioned in the data design dictionary are all part of my coding.

IDENTIFICATION OF STORAGE MEDIA

Since this is a non-database project, storage for a database is not required. However storage is still required for the actual application itself. Since this will be accessed by the Mathematics department during school time I believe the best medium for storage would be the school computers' Hard disk drives (HDDs). Since there are numerous computers throughout the school, copies of the application should only be stored on all those computers located in Mathematics Classrooms. I further believe that this would be the best method for storage because a HDD can be read from a multitude of times and has a relatively short access time of less than 10ms.

In terms of transferring medium (for transferring my application to other computers) , I think that a 256MB USB flash drive would be sufficient as it has more than enough capacity, is portable and is primarily used for transporting files and applications. The same USB memory stick could actually be used as a back-up, in case the application is accidently deleted from a computer. It would serve as a suitable back-up because it is portable (easy to store in a small safe), has a long lifetime and can directly run applications.

HARDWARE SPECIFICATION

- **Keyboard** – Used to input graph transformation magnitudes
- **Mouse** – Used to interact with forms
- **Visual Display Unit** (preferably LCD) – To display application
- **Hard disk drive (HDD)** – To store application
- **256MB USB drive** - Used for transferring application and as a back-up

5. IDENTIFICATION OF SUITABLE ALGORITMS

As you can see from my system flowcharts and structure charts, my project requires the production and use of a variety of algorithms. To begin with I will list the algorithms required:

- **SetXYAxesLimits** – This sets the minimum and maximum ranges of the x and y axes.
- **SetXYAxesIntervalsAndLayers** – This sets the X and Y Axes intervals, which will be labelled. I will explain what Layers are in the following section.
- **PlotXYAxes** – This plots the X and Y Axes on the graph
- **PlotFunction** – This plots the original function that is currently selected. There will nine different versions of this algorithm, as nine functions are required.
- **GetFunctionTransformationMagnitudes** - This reads in data from the sliders and tick boxes corresponding to Function transformations
- **ApplyTransformation** – This carries out all the different function transformations the user has selected onto the original graph and plots the transformed graph.
- **HighlightPoints** – This calculated the points to be highlighted on the original and transformed graphs, and plots them on top of the original and transformed graphs in a separate layer.
- **CalculateTransformedGraphInfo** – This calculates the mathematical description of function transformations currently applied, so it can be display to the user.
- **ShowGraphInfo** – This shows the mathematical description of the function transformations to the user.
- **CalcAndShowCartesianCoord** – This calculated and displays the Cartesian coordinates of the highlighted points.
- **OptionsBox** – This relates with all the different options that can be changed within the OptionsBox Form.
- **ResetGraph** – This resets all the function transformations back to their initialised values.
- **PrintGraph** – This deals with the all the different procedures involved in printing the graph

Before I continue to write the algorithms for each procedure, I feel it is necessary I give some information of the PlotPanel Component.

NB I will be quoting all the information from the website:
http://wiki.lazarus.freepascal.org/PlotPanel

PLOTPANEL

PlotPanel is a component for Lazarus to replace (more or less) the TChart component. With PlotPanel you can draw line, dot and bar graphs. Animated graphs are also possible. The PlotPanel component is derived from the TCustomPanel-component within Delphi. To this derived panel, plotting capabilities are added.

A lot of properties and methods are inherited from TCustomPanel. However the following properties and methods are added to PlotPanel:

BASIC PROPERTIES

Property	Description
BackColor	This is the background color of the plotting area
GridColor	
LayerOption	For easy plotting of a single layer this can be set to False. It is not possible to set this to True directly (see methods)
Margin	The width of the border of the plot in pixels
PlotMode	Three plotmodes are possible: pmBar, pmLine and pmDot
PlotPen	Properties of the plotting pen. If LayerOption is False, then all sub-properties can be set. If LayerOption is True, then information about the plotting pen is transferred via the LayerOptions method (methods)
Title	The Title is at the top center of the plot and uses the Font property
PlotBMP	This is a bitmap holding the whole graph (see demo for its use)

X-AXIS PROPERTIES

There is a similar set of properties for the Y-axis.

Property	Description
XLabel	Label at the X-axis
XMarks	When True, the X-axis has marks
XMarksFont	Font used for the XMarks
XMarksInterval	Interval between to lines in the grid (applies for linear scales only)
XMax	Maximum value of X that is visible
XMin	Minimal value of X that is visible
XScaleLog	When TRUE the scale is logarithmic (the interval is determined by the component)

METHODS

AddXY(X,Y: Extended) With this method you can add points to the plot, if there is more than one point then they are joint to make a line. E.g.

```
Var
  i: integer;

Begin
For i:= -10 to 10
  do PlotPanel1.AddXY(i,i*i) ;
End.
```

This plots a parabola .The line is plotted with the *PlotPen* properties. Note, you can only plot a single layer.

AddXY(X,Y: Extended; Color: Tcolor; Layer: Integer) A layer represents one graph area, adding more layers, creates further graph areas on top of the one/s already there. With this method it is possible to plot up to 8 lines in different colors, *plotmodes* and *penwidths* (see the **LayerOptions** method to see how to set *linewidth* and *plotmodes*)

LayerOptions(Layer: Integer; PlotMode: TPlotMode; PenWidth: Integer) Every Layer can use a different *Plotmode* and *PenWidth*. When you call this method, the *LayerOption* Boolean property will be set TRUE. When you want to use a single Layer again you have to set *LayerOption* to false.

Autoscale(Layer: Integer) With this method adjusts the size of the plot automatically so that the plotted function is displayed in the whole graph area.

ClearData Clears the data on the plot.

ConvertS2W(x,y: Extended ; var WX,WY : Extended) Converts the screen-coordinates in X,Y to the real coordinates WX,WY. When X and Y are out of bounds, the function returns False, otherwise True

Freeze(Boolean) When *Freeze(True)* is executed all plotting is done on a invisible bitmap. After a *Freeze(False)* the whole plot is displayed. This produces flicker-free animations.

HideLayer(Layer: Integer) Use this to hide a single layer in the plot.

UnHideLayer(Layer: Integer) Shows the hidden layer again N.B. When you add points to a hidden layer, the old layer is deleted and the new points are the only points in this layer.

Paint Repaints the whole Graph (is done automatically by the system when necessary)

I felt it was necessary to explain the way the PlotPanel component works so that my algorithms would be easier to follow and understand. Now that I have given a short explanation on the use of PlotPanel, I will begin writing my algorithms that I defined earlier:

NB* I will be using some variables from my design data dictionary. Also, anything enclosed in these { } brackets is a comment and not part of the algorithm.

SET XY AXES LIMITS

Var Xmin = minimum X-Axis Value; {Specified by programmer}
Var Xmax = maximum X-Axis Value; {Specified by programmer}
Var Ymin = minimum Y-Axis Value; {Specified by programmer}
Var Ymax = maximum Y-Axis Value; {Specified by programmer}

Set PlotPanel X-Axis range from **Xmin** to **Xmax;**
Set PlotPanel Y-Axis range from **Ymin** to **Ymax;**

SET XY AXES INTERVALS AND LAYERS

Var Xinter = X-Axis interval value; {Specified by programmer}
Var Yinter = Y-Axis interval value; {Specified by programmer}

Set PlotPanel X Axis intervals equal to **Xinter;**
Set PlotPanel Y Axis intervals equal to **Yinter;**

Create PlotPanel *layer* for X-Axis;
Create PlotPanel *layer* for Y-Axis;
Create PlotPanel *layer* for Original graph;
Create PlotPanel *layer* for Transformed graph;
Create PlotPanel *layer* for Inverse graph;
Create PlotPanel *layer* for highlighted point on Original graph;
Create PlotPanel *layer* for highlighted point on Transformed graph;

PLOT XY AXES

```
Var  Xmin = minimum X-Axis Value; {Specified by programmer}
Var  Xmax  = maximum X-Axis Value; {Specified by programmer}
Var  Ymin = minimum Y-Axis Value; {Specified by programmer}
Var Ymax = maximum Y-Axis Value ; {Specified by programmer}
Var count = 0;

For  count ← Xmin to Xmax
{
  Plot point on PlotPanel with coordinates of (count,0);
}
 {This produces a straight line which acts as the X-Axis}

For count ← Ymin to Ymax
{
  Plot point on PlotPanel with coordinates of (0,count);
}
{This produces a straight line which acts as the Y-Axis}
```

PLOT FUNCTION

Since this method will be classified as polymorphic later on when I document my class definitions, it won't have one set algorithm. The method will be implemented differently for each type of mathematical function used. So here I will try and write a general form of algorithm:

```
Var count = 0; {This will be a counter variable in the program, it is used to increment the value of x, so that it can be calculated for the whole plot}
Var YValue = y-value to be plot;
Var countmin = The minimum value of the count;
Var countmax = The maximum value of the count;

For count ← countmin to countmax
{
  YValue ← Transform count using mathematical expression corresponding to function;
  Plot point on PlotPanel with coordinates (count, YValue) ;
}
```

Each mathematical function will have its own **countmin** and **countmax** values depending on the number of points that need to be drawn. Also there might be more than one 'for loop' because certain functions have undefined regions (values of x where y equals infinity) and can't be drawn for those certain value of x, so we plot around those regions. In order to further explain this method I feel it necessary to give the algorithm for implementation of **PlotFunction** if the chosen function is quadratic:

```
Var count = 0; {This will be a counter variable in the program, it is used to increment the value of x, so that it can be calculated for the whole plot}
Var YValue = y-value to plot;
Var countmin = -1000;
Var countmax = 1000;

For count ← countmin to countmax
{
  YValue ← (0.01*count)*(0.01*count); { y = x}
  Plot point on PlotPanel with coordinates ((0.01*count), YValue);
}
```

As you can see **YValue** is calculated by squaring **count,** so when the 'for loop' is fully executed a graph of the quadratic function should be produced. For **countmin** and **countmax** I have used values 100 times larger than the **Xmin** and **Xmax** values of the graph because it allows the plotting of 2000 points in the same space rather than just 20. This increases the accuracy of the graph by decreasing the jaggedness of the overall quadratic curve.

GET FUNCTION TRANSFORMATION MAGNITUDES

```
Var a = Magnitude of Y-Translation transformation;
Var b = Magnitude of X-Translation transformation;
Var c = Magnitude of Y-Stretch transformation;
Var d = Magnitude of X-Stretch transformation;
Var e = Magnitude of X-Coordinate;
Var f = Y-Axis Reflection;
Var g =X-Axis Reflection;
Var InvX = Inverse Transformation;

Get values of a,b,c,d and e from graph transformation user interface form

If (f is selected in graph transformation user interface form)
  f ←-1;
Else
  f ← 1;

If (g is selected graph transformation user interface form)
  g ← -1;
Else
  g ← 1;

If (InvX is selected graph transformation user interface form)
  InvX ← true;
Else
  InvX ← false;
```

APPLY TRANSFORMATION

Like the **PlotFunction** method, this is also a polymorphic. It will be implemented differently by each class. Therefore the following will be a general algorithm:

```
Var a = Magnitude of Y-Translation transformation;
Var b = Magnitude of X-Translation transformation;
Var c = Magnitude of Y-Stretch transformation;
Var d = Magnitude of X-Stretch transformation;
Var f = Y-Axis Reflection;
Var g =X-Axis Reflection;
Var InvX = Inverse Transformation
Var = count = 0;
Var YValue = y-value to plot;
Var countmin = The minimum value of the count;
Var countmax = The maximum value of the count;

For count ← countmin to countmax
{
  YValue ← Transform count using transformed mathematical expression of function
involving a,b,c,d,f and g;
  Plot point on PlotPanel with coordinates (count, YValue);
}

If InvX <-- true
{
  Hide all Layers currently visible except X and Y Axes;
  For count ← countmin to countmax
    {
    YValue ← Transform count using inverse mathematical expression of function;
    Plot point of PlotPanel with coordinates (count, YValue) on Layer created for the inverse
function;
    }
  Show Layer with plot of inverse function;
}
```

Each mathematical function will have its own **countmin** and **countmax** values depending on the number of points that need to be plot. Also there might be more than one 'for loop' because certain functions have undefined regions and can't be plot for those certain value of x, so we plot around those regions. In order to further explain this I feel it necessary to give the algorithm for implementation of **ApplyFunctionTransformations** if the chosen function is linear (i.e. straight line graph):

```
Var a = Magnitude of Y-Translation transformation;
Var b = Magnitude of X-Translation transformation;
Var c = Magnitude of Y-Stretch transformation;
Var d = Magnitude of X-Stretch transformation;
Var f = Y-Axis Reflection;
Var g =X-Axis Reflection;
Var InvX = Inverse Transformation
Var = count = 0;
Var YValue = y-value to plot (Linear);
Var countmin = -10;
Var countmax = 10;

For count ← countmin to countmax
{
  YValue ← f*g*((d*(c*(count+b)))+a);
  Plot point on PlotPanel with coordinates (count, YValue);
}

If InvX <-- true
{
  Hide all Layers currently visible except X and Y Axes;
  For count ← -1000 to -1
  {
    YValue ← (1/(0.01*count));
    Plot point of PlotPanel with coordinates (count, YValue) on different Layer;
  }
  For count ← 1 to 1000
  {
    YValue ← (1/(0.01*count));
    Plot point on PlotPanel with coordinates (count, YValue) on different Layer;
  }
  Show Layer with plot of inverse function;
}
```

As you can see for the second graph, where **InvX** is true, there are two 'for loops' because the value of **count** = 0 has to be left out. If it was included, **YValue** would equal 1 divided 0, which would cause an error and cause the program to crash.

HIGHLIGHT POINTS

This is also a polymorphic method, here is the general algorithm:

Var a = Magnitude of Y-Translation transformation;
Var b = Magnitude of X-Translation transformation;
Var c = Magnitude of Y-Stretch transformation;
Var d = Magnitude of X-Stretch transformation;
Var e = Magnitude of X-Coordinate
Var f = Y-Axis Reflection;
Var g =X-Axis Reflection;
Var InvX = Inverse Transformation
Var YValue = y-value to plot;

If InvX = false
{
Hide layers created for highlighted points {so that we can plot on them without any flicker appearing on the graph}
YValue ← Transform *count* using transformed mathematical expression of function involving use of **a,b,c,d,e,f** and **g**
Plot point on PlotPanel with coordinates **(e, YValue)** on *layer* assigned for "highlighted point on transformed graph"
Plot point on PlotPanel with coordinates **(e, (**selected function applied to **e))** on *layer* assigned for "highlighted point on Original graph"
}

If we were to look at the implementation algorithm of this method for the linear function, the changes would be the following: (changes highlighted)

If InvX = false
{
Hide layers created for highlighted points {so that we can plot on them without any flicker appearing on the graph}
YValue ← f *g*((d*(c*(e+b)))+a)
Plot point on PlotPanel with coordinates **(e, YValue)** on *layer* assigned for "highlighted point on transformed graph"
Plot point on PlotPanel with coordinates **(e, e)** on *layer* assigned for "highlighted point on Original graph"
}

CALCULATE TRANSFORMED GRAPH INFO

This will also be a polymorphic method

```
Var a = Magnitude of Y-Translation transformation;
Var b = Magnitude of X-Translation transformation;
Var c = Magnitude of Y-Stretch transformation;
Var d = Magnitude of X-Stretch transformation;
Var e = Magnitude of X-Coordinate
Var f = Y-Axis Reflection;
Var g =X-Axis Reflection;
Var  InvX = Inverse Transformation;
Var TextInfo = String displaying information about transformed graph;
Var Sa ← Convert a to text;
Var Sb ←Convert b to text;
Var Sc ← Convert c to text;
Var Sd ← Convert d to text;

If (f ← -1)
{
  If ( g ← -1)
  {
    TextInfo ← Incorporate Sa,Sb,Sc and Sd into general text  describing the transformation
          and that the graph has been reflected in lines through the X and Y axis;
  }
  Else
  {
    TextInfo ← Incorporate Sa,Sb,Sc and Sd into general text  describing the transformation
          and that the graph has been reflected in a line through the X axis;
  }
}
Else
{
  If ( g ← -1)
  {
    TextInfo ← Incorporate Sa,Sb,Sc and Sd into general text  describing the transformation
          and that the graph has been reflected in a line through the Y axis;
  }
  Else
  {
    TextInfo ← Incorporate Sa,Sb,Sc and Sd into general text  describing the transformation;
  }
}
```

SHOW GRAPH INFO

This will be a polymorphic method, since each function has a different description.

Var TextInfo = String displaying information about transformed graph;

If (***InvX*** ← true)
{
  Output text saying that the inverse of the currently selected function has been drawn;
}

Else
{
  Output text indicating which function has been drawn;
  Output ***TextInfo*** ;
}

CALC AND SHOW CART COORD

This is a Polymorphic method. The following is a general algorithm:

Var a = Magnitude of Y-Translation transformation;
Var b = Magnitude of X-Translation transformation;
Var c = Magnitude of Y-Stretch transformation;
Var d = Magnitude of X-Stretch transformation;
Var e = Magnitude of X-Coordinate
Var f = Y-Axis Reflection;
Var g =X-Axis Reflection;
Var InvX = Inverse Transformation;
Var XRed = The value of the Cartesian coordinate corresponding to point highlighted on the original graph
Var YRed = The value of the Cartesian coordinate corresponding to point highlighted on the original graph
Var XBlue = The value of the Cartesian coordinate corresponding to point highlighted on the transformed graph
Var YBlue = The value of the Cartesian coordinate corresponding to point highlighted on the transformed graph

If (***InvX*** = false)
{
  XRed ← **e;**
  XBlue ← **e;**
  YRed ← **Calculate** the result of the function currently selected performed on **e;**
  YBlue ← **Caluclate** the combined result of function currently selected and its transformations performed on **e;**
  Output Cartesian coordinates in the following form **(XRed,YRed)** and **(XBlue, YBlue)**;

```
}
```

If we were to look at the implementation algorithm of this method for the Quadratic function, the changes would be the following: (changes highlighted in yellow)

```
If (InvX = false)
{
  XRed ← e;
  XBlue ← e;
  YRed ← e*e;
  YBlue ← f *(c*((g*(d*((e)+b)))*(g*(d*((e)+b))))+a);
 Output Cartesian coordinates in the following form (XRed,YRed) and (XBlue, YBlue);
}
```

OPTIONS BOX

(Refer to GUI design on page 34 of Design)

```
Create Options box Form

If (Options Button on Main form clicked)
    Set Options box Form to visible;

If (Tickbox corresponding to XLabel is checked)
    Set XMarks on PlotPanel to true;
Else
    Set XMarks on PlotPanel to false;

If (Tickbox corresponding to YLabel is checked0
    Set YMarks on PlotPanel to true;
Else
    Set YMarks on PlotPanel to false;

If (Tickbox corresponding to GridLines is checked)
    Set Grid Color on PlotPanel to Black;
Else
    PlotPanel Grid Colour ← PlotPanel Back Colour;

If (BackColour colour selector changes)
    PlotPanel Back Colour ← Colour currently selected;

If (Page setup clicked)
   Execute PageSetup dialog;
```

RESET GRAPH

```
If (Reset Button clicked on Main Form)
{
  Intialise slider positions
  Intialise text boxes
  Intialise Check boxes
  Update PlotPanel and TextArea containing graph information
}
```

PRINT GRAPH

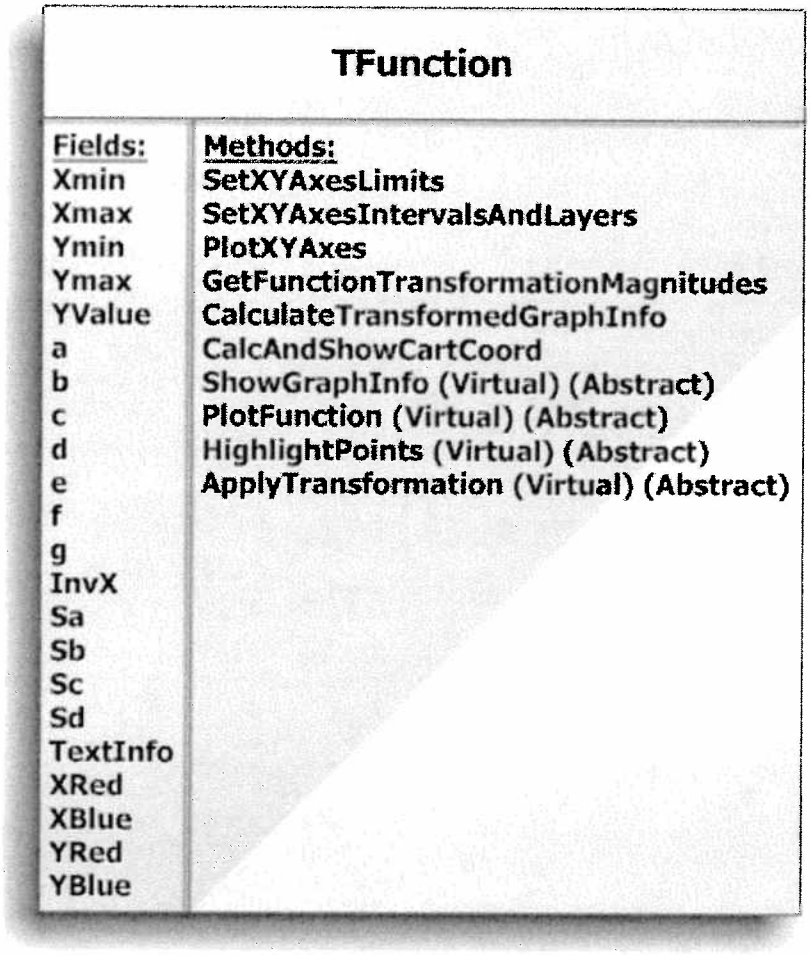
```
If (Print Button clicked on Main Form)
{
Create PrinterDialog;

If (PrinterDialog executed)
 {
  No. Of Copies ← Copy Number specified in Printer Dialog
  Print Bitmap image of PlotPanel Graph Area
 }
```

6. CLASS DIAGRAMS, OBJECT BEHAVIOURS AND METHODS

CLASS DIAGRAMS

BASE CLASS: TFUNCTION



By **Virtual** I mean that the methods are polymorphic and can be redefined in any further sub-classes of TFunction.

I have also defined some methods as **Abstract**, meaning that they are not implemented by the base class but are later defined and used by its sub-classes. This means I don't have to code the methods for the base class but will have to for any of its subclasses.

See http://www.delphibasics.co.uk/Article.asp?Name=Abstract

SUB CLASSES OF TFUNCTION

TLinear

Inherited Fields:	Inherited Methods:
Xmin	SetXYAxesLimits
Xmax	SetXYAxesIntervalsAndLayers
Ymin	PlotXYAxes
Ymax	GetFunctionTransformationMagnitudes
YValue	CalculateTransformedGraphInfo
a	CalcAndShowCartCoord
b	ShowGraphInfo (Override)
c	PlotFunction (Override)
d	HighlightPoints (Override)
e	ApplyTransformation (Override)
f	
g	
InvX	
Sa	
Sb	
Sd	
Sd	
TextInfo	
XRed	
XBlue	
YRed	
YBlue	

Method that will implemented differently
ShowGraphInfo
PlotFunction
HighlightPoints
ApplyTransformation

TQuadratic

Inherited Fields:	Inherited Methods:
Xmin	SetXYAxesLimits
Xmax	SetXYAxesIntervalsAndLayers
Ymin	PlotXYAxes
Ymax	GetFunctionTransformationMagnitudes
YValue	CalculateTransformedGraphInfo
a	CalcAndShowCartCoord
b	ShowGraphInfo (Override)
c	PlotFunction (Override)
d	HighlightPoints (Override)
e	ApplyTransformation (Override)
f	
g	
InvX	
Sa	
Sb	
Sc	
Sd	
TextInfo	
XRed	
XBlue	
YRed	
YBlue	

Methods that will implemented differently
ShowGraphInfo
PlotFunction
HighlightPoints
ApplyTransformation

TCubic

Inherited Fields:	Inherited Methods:
Xmin	SetXYAxesLimits
Xmax	SetXYAxesIntervalsAndLayers
Ymin	PlotXYAxes
Ymax	GetFunctionTransformationMagnitudes
YValue	CalculateTransformedGraphInfo
a	CalcAndShowCartCoord
b	ShowGraphInfo (Override)
c	PlotFunction (Override)
d	HighlightPoints (Override)
e	ApplyTransformation (Override)
f	
g	
InvX	
Sa	
Sb	
Sc	
Sd	
TextInfo	
XRed	
XBlue	
YRed	
YBlue	

Methods that will implemented differently
ShowGraphInfo
PlotFunction
HighlightPoints
ApplyTransformation

TSine

Inherited Fields:	Inherited Methods:
Xmin	SetXYAxesLimits
Xmax	SetXYAxesIntervalsAndLayers
Ymin	PlotXYAxes
Ymax	GetFunctionTransformationMagnitudes
YValue	CalculateTransformedGraphInfo
a	CalcAndShowCartCoord
b	ShowGraphInfo (Override)
c	PlotFunction (Override)
d	HighlightPoints (Override)
e	ApplyTransformation (Override)
f	
g	
InvX	
Sa	
Sb	
Sc	
Sd	
TextInfo	
XRed	
XBlue	
YRed	
YBlue	

Methods that will implemented differently
ShowGraphInfo
PlotFunction
HighlightPoints
ApplyTransformation

TCosine

Inherited Fields:	Inherited Methods:
Xmin	SetXYAxesLimits
Xmax	SetXYAxesIntervalsAndLayers
Ymin	PlotXYAxes
Ymax	GetFunctionTransformationMagnitudes
YValue	CalculateTransformedGraphInfo
a	CalcAndShowCartCoord
b	ShowGraphInfo (Override)
c	PlotFunction (Override)
d	HighlightPoints (Override)
e	ApplyTransformation (Override)
f	
g	
InvX	
Sa	
Sb	
Sc	
Sd	
TextInfo	
XRed	
XBlue	
YRed	
YBlue	

Methods that will implemented differently
ShowGraphInfo
PlotFunction
HighlightPoints
ApplyTransformation

TTangent

Inherited Fields:	Inherited Methods:
Xmin	SetXYAxesLimits
Xmax	SetXYAxesIntervalsAndLayers
Ymin	PlotXYAxes
Ymax	GetFunctionTransformationMagnitudes
YValue	CalculateTransformedGraphInfo
a	CalcAndShowCartCoord
b	ShowGraphInfo (Override)
c	PlotFunction (Override)
d	HighlightPoints (Override)
e	ApplyTransformation (Override)
f	
g	
InvX	
Sa	
Sb	
Sc	
Sd	
TextInfo	
XRed	
XBlue	
YRed	
YBlue	

Methods that will implemented differently
ShowGraphInfo
PlotFunction
HighlightPoints
ApplyTransformation

TExponential

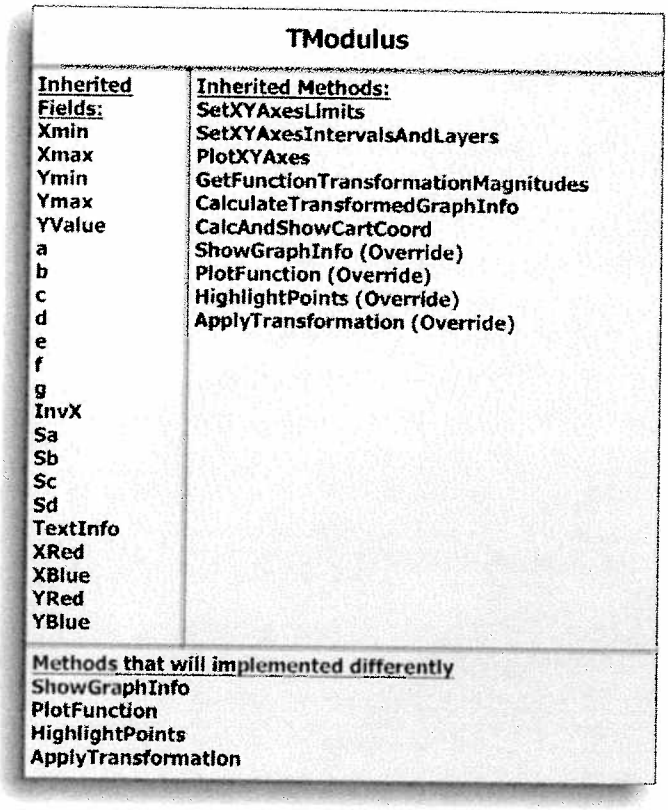
Inherited Fields:	Inherited Methods:
Xmin	SetXYAxesLimits
Xmax	SetXYAxesIntervalsAndLayers
Ymin	PlotXYAxes
Ymax	GetFunctionTransformationMagnitudes
YValue	CalculateTransformedGraphInfo
a	CalcAndShowCartCoord
b	ShowGraphInfo (Override)
c	PlotFunction (Override)
d	HighlightPoints (Override)
e	ApplyTransformation (Override)
f	
g	
InvX	
Sa	
Sb	
Sc	
Sd	
TextInfo	
XRed	
XBlue	
YRed	
YBlue	

Methods that will implemented differently
ShowGraphInfo
PlotFunction
HighlightPoints
ApplyTransformation

TInverse

Inherited Fields:	Inherited Methods:
Xmin	SetXYAxesLimits
Xmax	SetXYAxesIntervalsAndLayers
Ymin	PlotXYAxes
Ymax	GetFunctionTransformationMagnitudes
YValue	CalculateTransformedGraphInfo
a	CalcAndShowCartCoord
b	ShowGraphInfo (Override)
c	PlotFunction (Override)
d	HighlightPoints (Override)
e	ApplyTransformation (Override)
f	
g	
InvX	
Sa	
Sb	
Sc	
Sd	
TextInfo	
XRed	
XBlue	
YRed	
YBlue	

Methods that will implemented differently
ShowGraphInfo
PlotFunction
HighlightPoints
ApplyTransformation



CLASS DEFINITIONS (THESE MAY CHANGE DURING IMPLEMENTATION STAGE)

TFUNCTION

```
TFunction = class
  Private
    Xmin : integer;
    Xmax : integer;
    Ymin : integer;
    Ymax : integer;
    a : integer;
    b : integer;
    c : integer;
    d : integer;
    e : integer;
    f  : integer;
    g : integer;
    InvX : boolean;
    YValue : Extended;
    Sa, Sb, Sc, Sd  : string;
    TextInfo : string;
    XRed, XBlue, YRed, YBlue  : string;
```

```
  Public
    Procedure SetXYAxesLimits;
    Procedure SetXYAxesIntervalsAndLayers;
    Procedure PlotXYAxes;
    Procedure PlotFunction; Virtual; Abstract;
    Procedure GetFunctionTransformationMagnitudes;
    Procedure ApplyTransformation; Virtual; Abstract;
    Procedure HighlightPoints; Virtual; Abstract;
    Procedure CalculateTransformedGraphInfo;
    Procedure ShowGraphInfo;
    Procedure CalcAndShowCartCoord; Virtual; Abstract;
End;
```

SUB CLASSES OF TFUNCTION

For example, here is **TLinear**:

```
TLinear = class (TFunction)
  Procedure PlotFunction; Override;
  Procedure ApplyTransformation; Override;
  Procedure HighlightPoints; Override;
  Procedure CalcAndShowCartCoord; Override;
End;
```

TQuadratic, TCubic, TSine, TCosine, TTangent, TExponential, TInverse and **TModulus** are all similar to **TLinear** in terms of class definition, the only difference being their names.

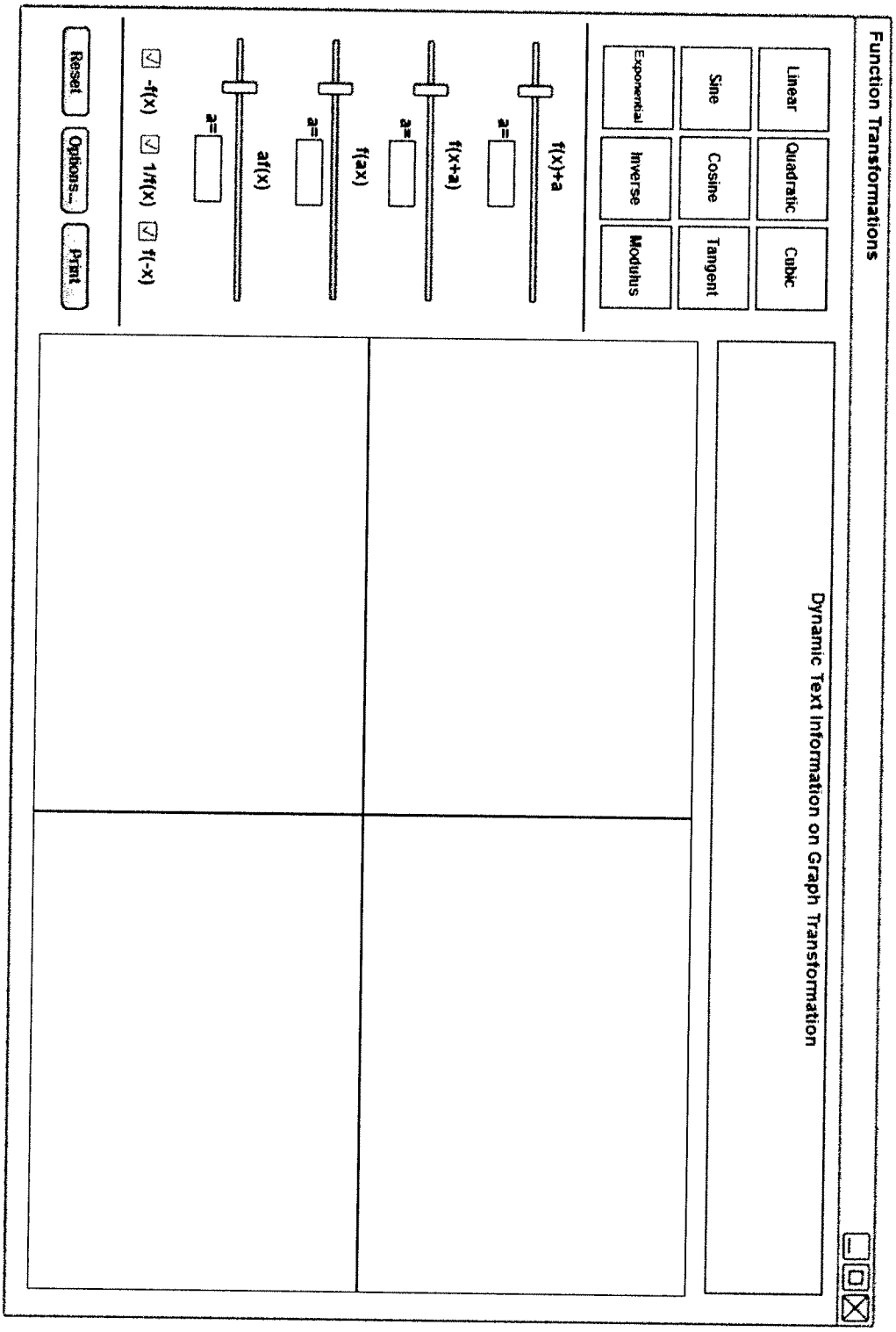
7. USER INTEFACE DESIGN

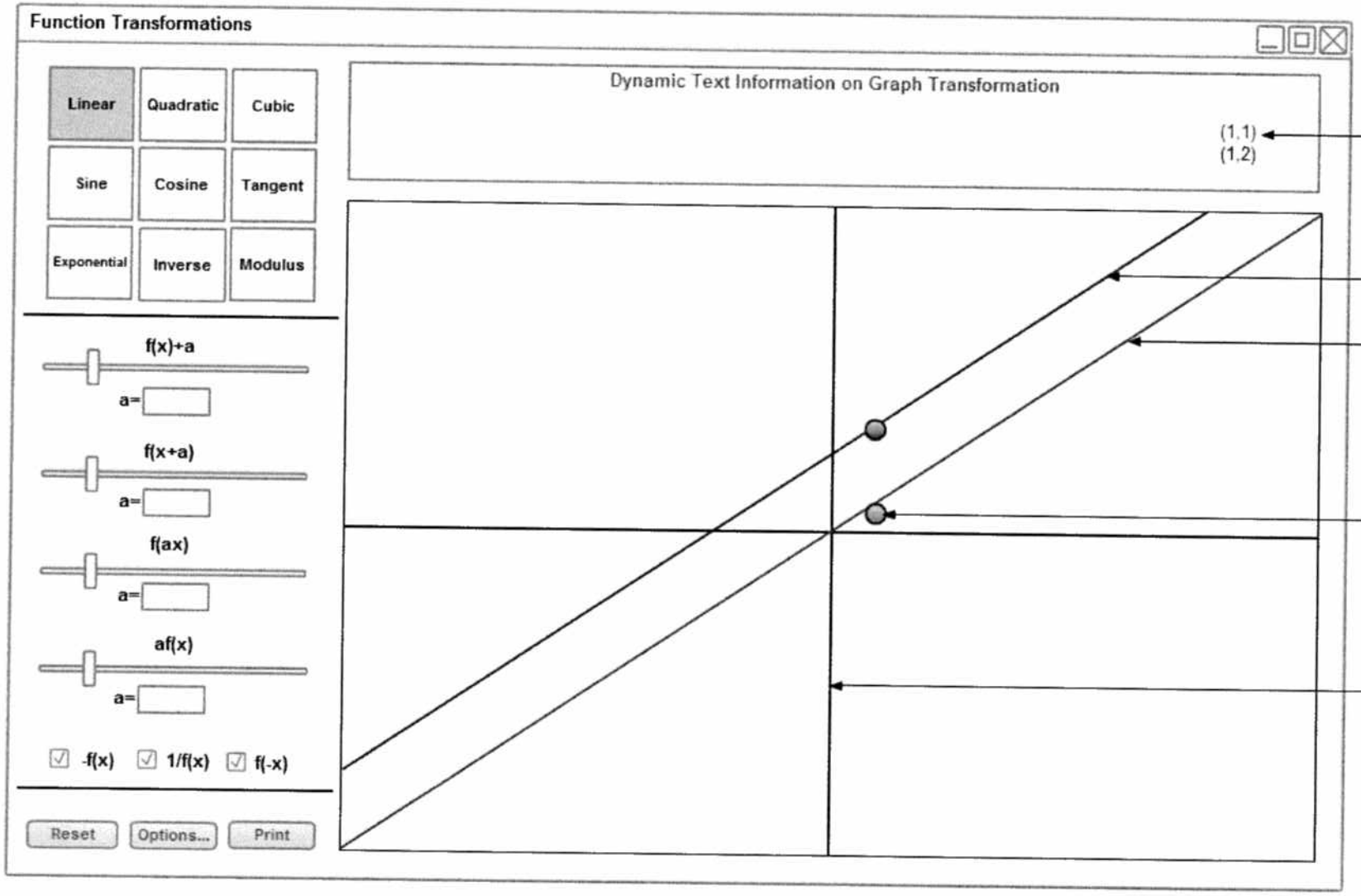
UI SAMPLE OF PLANNED INPUT WITH OUTPUT DESIGNS

These prototypes have been designed using online diagram software called Gliffy.

Website: http://www.gliffy.com/

GRAPH TRANSFORMATION FORM




Function Transformations
Linear
Quadratic
Cubic
Sine
Cosine
Tangent
Exponential
Inverse
Modulus
f(x)+a
a=
f(x+a)
a=
f(ax)
a=
af(x)
a=
-f(x)
1/f(x)
f(-x)
Reset
Options...
Print
Dynamic Text Information on Graph Transformation
(1,1)
(1,2)
Cartesian Coordinates of highlighted points
Transformed graph
Original graph
Highlighed point
Y axis

For function selection I chose a collection of toggle boxes shaped together in such a way so that they resemble a keypad. This decreases the amount of hand mobility required when moving from different functions. The larger surface area of the toggle boxes also makes it easier to select the functions in the first place.

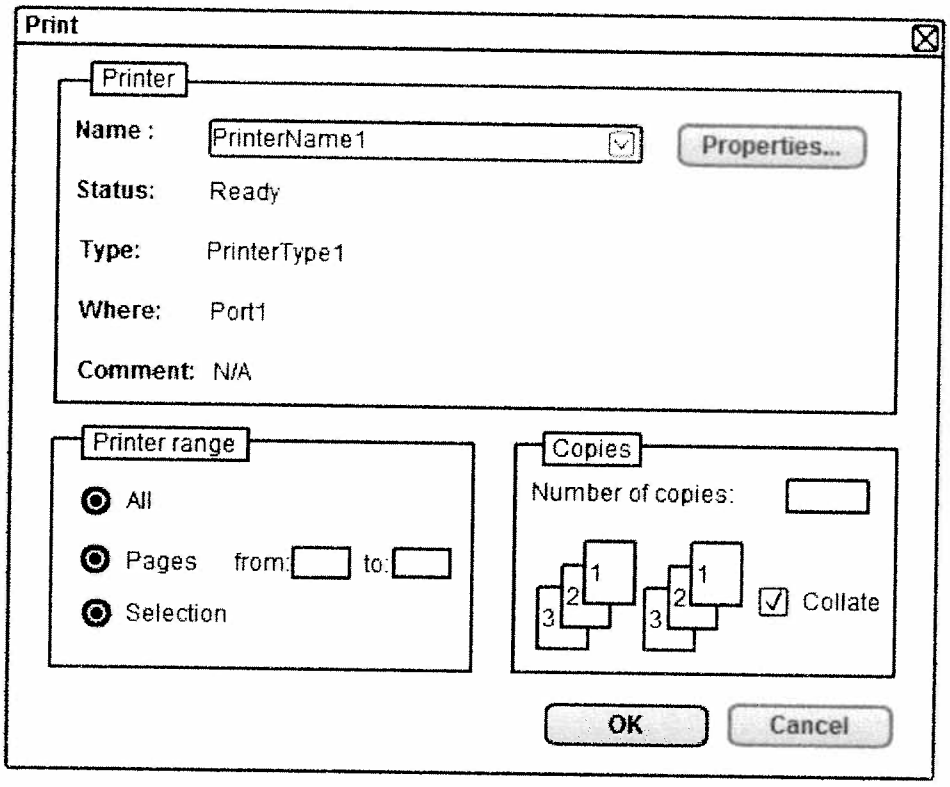
To vary the magnitude of a function transformation I used track-bars ('sliders') coupled with textboxes. The track bars allow for a smooth variation of the magnitudes, and I included the textboxes for accuracy so that the user could enter a specific value if required. To select a function transformation without a magnitude I chose tick boxes because they can only be in one of two states, checked or unchecked. These states correspond to the transformation states as it can be applied or not applied.

I made the plot area of the graph as large as possible so that all the Maths students could easily see the graph from a screen projector, no matter where they were seated in the classroom. The fact that I will use different colours to represent the original and transformed plots also allows the graph transformations to be seen more easily. In addition, I placed the text holder, displaying information about the graphs, directly above the plot area in order to emphasize it since it relates to what exactly is happening to the graph (due to the effects of the transformations).

I have also used normal buttons for print selection, resetting the graph and opening up the options box. As you can see I have actually used conventional components (such as sliders, tick boxes, buttons, etc) throughout the form in order to make it easy to use even for the novice user. This form complies with the full set of user requirements.

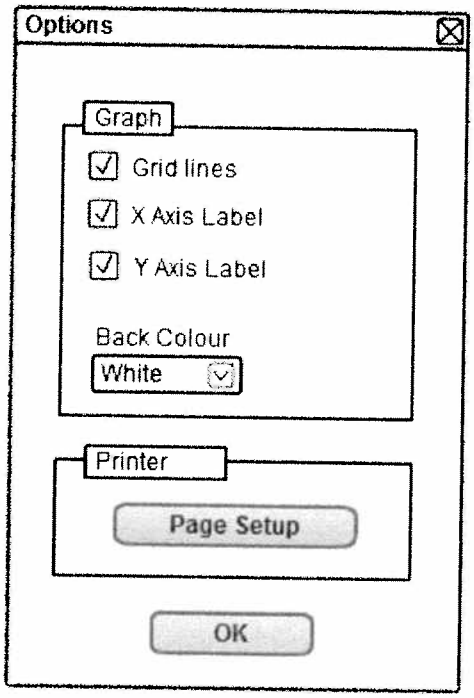
UI SAMPLE OF PLANNED DATA CAPTURE AND ENTRY DESIGNS

PRINTER DIALOG (WITHIN MICROSOFT WINDOWS)



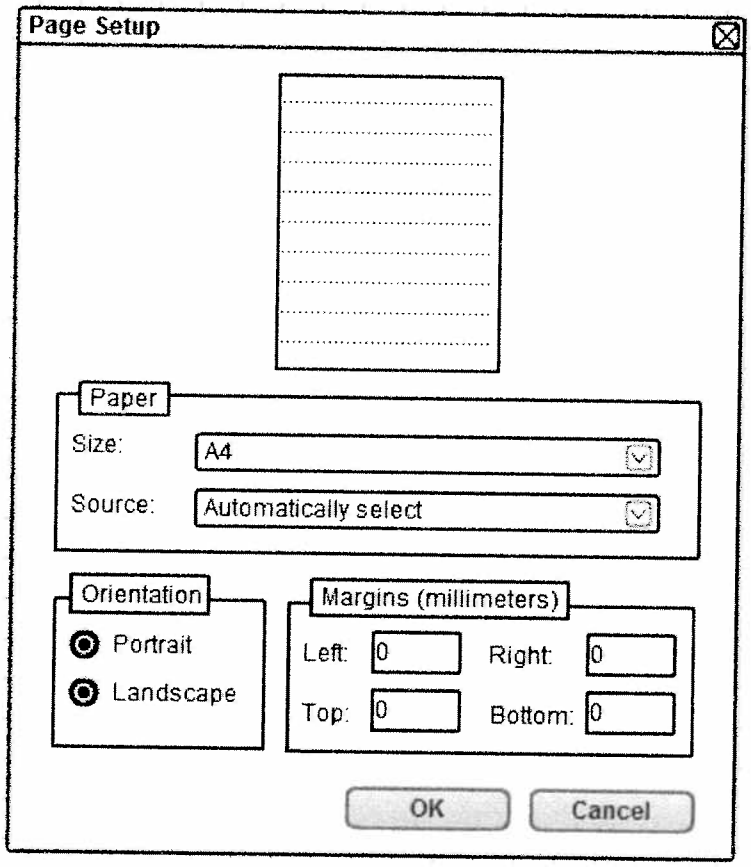
I have not designed this dialog; it is part of the standard Microsoft Windows GUI. I am just using it because it is available in the compiler and is the default dialog for the task I wish to carry out. It also explains the information clearly to the user.

OPTIONS BOX DIALOG



This form allows for easy customisation of the program, with the use of conventional tick boxes, buttons and a colour selector. Like the previous form, this dialog also matches the user level sufficiently and allows for a clear, transparent user interface. For instance, the exit is clearly marked with the close button and labels have been used to show which options apply to the graph and which apply to the printer.

PAGE SETUP DIALOG (WITHIN MICROSOFT WINDOWS)



I have not designed this dialog; it is part of the standard Microsoft Windows GUI. I am just using it because it is available in the compiler and is the default dialog for the task I wish to carry out. It also explains the information clearly to the user.

USER INTERFACE DESIGN RATIONALE

I chose a Graphical User Interface (GUI) over a Command Line Interface (CLI) because my project mainly involves the presentation and manipulation of graphical information, which would only be possible through a GUI. For instance, a GUI allows the use of different dialogs to display different parts of my program and the use of a pointer to select objects and commands. In addition, it suits the user's ability as they don't have to have knowledge of complex command languages needed to operate a CLI.

8. SECURITY AND INTEGRITY OF DATA

Since my project is a non-database project, data security is not a concern for it. However the integrity and correctness of data is quite essential. I have mentioned some of the data validation required for my object fields in my data design dictionary.

The magnitude of any function transformation (for those that actually have one) can be input in two different ways: by varying a slider to the correct position or by entering the value of the magnitude into a textbox. The slider bar has set positions representing integers along its length, and the user can only move the slider to those exact positions so data input this way will always be valid. On the other hand, the text box can take any character from the keyboard as input, even if what we require is an integer. Fortunately, the component that I will use is specially designed by the IDE so that it automatically validates data; it prevents the user from entering any erroneous data such as characters instead of integers. The only other methods of input are the pressing of buttons, toggle-boxes and check-boxes, which don't require any validation since they have two set states. Therefore I can conclude that my data integrity is sufficient for my project.

9. SYSTEM SECURITY

The only users for my project will be teachers from the Mathematics department, meaning the application won't be open to students. Therefore, it is a system security requirement that this application is only available to teachers. I believe that this requirement can be established by the school system administrator.

Each Mathematics classroom is only equipped with one computer which is mainly used by the teacher. Therefore, the administrator only needs to make my application available on these computers. However, since students still have the ability to log in to these computers, the administrator will also have to ensure that the application is only visible to teacher accounts on the system. Another alternative is making the application only available to Mathematics teachers wherever they are working, regardless of the computer.

10. TEST STRATEGY

It is extremely important that I test my program, not to show that my program is working correctly but to reveal the presence on any errors present within the program. Before I actually begin the Testing phase of my write-up I believe it is necessary that I devise a strategy to organise and plan my testing in order to make it as efficient and thorough as it can be.

I have decided to go with the White-box or structural testing strategies, where the full coding will be available for me to test. In terms of types, I have chosen to implement Unit Testing followed by Integration Testing and finally, System testing. I may or may not do Acceptance testing; it really depends on the time constraints of my project.

Testing

Specification Excerpt

- Test the solution and document the evidence of testing
- Consider the different types of testing that may be applied to the developed system
- Justify the methods chosen to test the solution

TESTING CONTENTS

1. Detailed test plan 3

Unit testing 3

Integration testing 5

System testing 5

2. Minimal Test data 6

Unit testing 6

Integration testing 8

System testing 8

3. Unit testing 8

4. Integration testing 8

5. System testing 8

6. Test evaluation 8

1. DETAILED TEST PLAN

UNIT TESTING

FUNCTION TRANSFORMATIONS FORM

Test number	Purpose	Test data reference	Evidence reference
1	To check if X Axis range is correctly set	TD1	
2	To check if Y Axis range is correctly set	TD2	
3	To check if X-Axis is plot correctly	TD3	
4	To check if Y-Axis is plot correctly	TD4	
5	To check if y-coordinate is correctly calculated from x-coordinate for original graph of Linear function	TD5	
6	To check if y-coordinate is correctly calculated from x-coordinate for original graph of Quadratic function	TD6	
7	To check if y-coordinate is correctly calculated from x-coordinate for original graph of Cubic function	TD7	
8	To check if y-coordinate is correctly calculated from x-coordinate for original graph of Sine function	TD8	
9	To check if y-coordinate is correctly calculated from x-coordinate for original graph of Cosine function	TD9	
10	To check if y-coordinate is correctly calculated from x-coordinate for original graph of Tangent function	TD10	
11	To check if y-coordinate is correctly calculated from x-coordinate for original graph of Exponential function	TD11	
12	To check if y-coordinate is correctly calculated from x-coordinate for original graph of Inverse function	TD12	
13	To check if y-coordinate is correctly calculated from x-coordinate for original graph of Modulus function	TD13	
14	To check Y translation transformation is calculated correctly for a point	TD14	
15	To check X translation transformation is calculated correctly for a point	TD15	
16	To check Y stretch transformation is calculated correctly for a point	TD16	
17	To check X stretch transformation is calculated correctly for a point	TD17	

18	To check reflection in x-axis transformation is calculated correctly for a point	TD18	
19	To check reflection in y-axis transformation is calculated correctly for a point	TD19	
20	To check inverse transformation is calculated correctly for a point	TD20	
21	To check if point is correctly highlighted on Original graph	TD21	
22	To check if Cartesian coordinates of highlighted point on original graph are calculated correctly	TD22	
23	To check if point is correctly highlighted on transformed graph	TD23	
24	To check if Cartesian coordinates of highlighted point on transformed graph are calculated correctly	TD24	
25	Print Dialog appears on click of print button	TD25	
26	Clicking the Reset Button initialises graph	TD26	
27	Options box appears on click of options button	TD27	

OPTIONS BOX FORM

Test number	**Purpose**	**Test data**	**Evidence reference**
28	To check if 'X-Axis label' option functions correctly	TD28	
29	To check if 'Y-Axis label' option functions correctly	TD29	
30	To check if 'Grid' option functions correctly	TD30	
31	To check if Graph colour change is registered by 'Function Transformation' Form	TD31	
32	To check if Page Setup dialog correctly appears on click of 'Page Setup button'	TD32	

INTEGRATION TESTING

Test number	Purpose	Test data	Evidence reference
33	To check if X and Y Axes ranges are set properly on same graph	TD33	
34	To check if X and Y Axes plot correctly on same graph	TD34	
35	To check if all the points needed to plot the original graph of the function are all correctly calculated and displayed for the selected Function	TD35	
36	To check if Y translation transformation is calculated correctly for an entire graph	TD36	
37	To check if X translation transformation is calculated correctly for an entire graph	TD37	
38	To check if Y Stretch transformation is calculated correctly for an entire graph	TD38	
39	To check if X Stretch transformation is calculated correctly for an entire graph	TD39	
40	To check if reflection in x-axis transformation is calculated correctly for an entire graph	TD40	
41	To check if reflection in y-axis transformation is calculated correctly for an entire graph	TD41	
42	To check if inverse transformation is calculated correctly for an entire graph	TD42	
43	To check if points are highlighted simultaneously on original and transformed graphs	TD43	
44	To check if Cartesian coordinates are displayed for both original and transformed graphs	TD44	

SYSTEM TESTING

Test number	Purpose	Test data	Evidence reference
45	Graph transformation form test	TD45	
46	Options box test	TD46	
47	Full system test	TD47	

2. MINIMAL TEST DATA

UNIT TESTING

Test data reference	Test Data	Reason for choice	Expected result	Actual result
TD1	Xmin =-10 and Xmax = 10 Xinterval =1	Normal data	X-Axis Labelled correctly	X-Axis Labelled correctly
	Xmin =-360 and Xmax = 360 Xinterval =90	Normal data	X-Axis Labelled correctly	X-Axis Labelled correctly
TD2	Ymin =-10 and Ymax = 10 Yinterval = 1	Normal data	Y-Axis Labelled correctly	Y-Axis Labelled correctly
	Ymin =-5 and Ymax = 5 Yinterval = 1	Normal data	Y-Axis Labelled correctly	Y-Axis Labelled correctly
TD3	Xmin =-10 and Xmax = 10	Normal data	X Axis plot correctly (20 points plot to form a horizontal line)	X Axis plot correctly (20 points plot to form a vertical line)
TD4	Ymin =-10 and Ymax = 10 20 points plot to form a line	Normal data	Y Axis plot correctly (20 points plot to form a vertical line)	Y Axis plot correctly (20 points plot to form a vertical line)
TD5	X = 5	Normal data	Y = 5	Y = 5
	X = -10	Boundary data	Y = -10	Y = -10
TD6	X = -3	Normal data	Y = 9	
	X = 10	Boundary data	Y = 100 (but since Ymax = 10, it won't be visible on graph)	Not visible on graph
TD7	X =2	Normal data	Y = 8	Y = 8
	X = -1	Normal data	Y = -1	Y = -1
TD8	X = 90	Normal data	Y = 1	Y = 1
	X = -360	Boundary data	Y = 0	Y = 0
TD9	X = 360	Boundary data	Y = 1	Y = 1
	X = 90	Normal data	Y = 0	Y = 0
TD10	X = 45	Normal data	Y = 1	Y = 1
	X = -360	Boundary data	Y = 0	Y = 0
	X = 90	Erroneous data (asymptote)	Y should not be calculated for this point as it will give infinity and the program will crash, so a point near X = 90 should be calculated	Y = tan(89.9)

TD11	X = 0	Normal data	Y = 1	Y = 1
TD12	X= 1	Normal data	Y = 1	Y = 1
	X = -10	Boundary data	Y = -0.1	Y = -0.1 (slightly hard to see)
	X = 0	Erroneous data	Y should not be calculated for this point as it will give infinity and the program will crash, so a point near X = 0 should be calculated	Y= 1/0.01 = 100
TD13	X = -6	Normal data	Y = 6	Y = 6
	X = 10	Boundary data	Y =10	Y =10
TD14	Function=linear, X =0, Magnitude = 10	Normal data	Y = 10	Y = 10
	Function = Sine, X =360 Magnitude = 1	Normal	Y = 1	Y = 1
	Function = Inverse, X = 0	Erroneous data	Y should not be calculated for this point as it will give infinity and the program will crash, so a point near X = 0 should be calculated	Y= 1/0.01 = 100
TD15	Function = Quadratic, X= 2 Magnitude = 2	Normal data	Y = 6	Y = 6
	Function = Cosine, X = -270 Magnitude = -5	Boundary data	Y = -4	Y = -4
TD16				
TD17				
TD18				
TD19				
TD20				
TD21				
TD22				

TD23				
TD24				
TD25				
TD26				
TD27				

INTEGRATION TESTING

Due to Time constraints this section couldn't be completed

SYSTEM TESTING

Due to Time constraints this section couldn't be completed

3. UNIT TESTING

Due to Time constraints this section couldn't be completed

4. INTEGRATION TESTING

Due to Time constraints this section couldn't be completed

5. SYSTEM TESTING

Due to Time constraints this section couldn't be completed

6. TEST EVALUATION

Due to Time constraints this section couldn't be completed

Maintenance

Specification excerpt

- Develop and document a solution for maintainability
- Consider the factors that affect the maintainability of a solution and evaluate a solution for maintainability in terms of the ease with which a program/solution can be corrected if an error is encountered, adapted if its environment changes, or enhanced if the customer changes requirements

MAINTENANCE CONTENTS

1. System overview 3

2. Acknowledgement of non-candidate coding 4

3. Detailed algorithm design 5

Set Xy Axes limits 5

set xy axes intervals and layers 5

plot xy axes 6

get function transformation magnitudes 6

plot function 7

apply transformation 10

highlight points 16

options box 20

reset graph 22

print graph 22

4. procedure/Method and variable lists 24

Procedures/methods 24

Variable/Field lists 26

5. ANNOTATED code listing 28

6. ANNOTATED design views of self created forms 28

Form1 (Main Form) 28

OptionsBox 29

1. SYSTEM OVERVIEW

I have created a system containing four forms/dialogs, which allows the user to carry and view the transformations of the graphs of nine different functions. It also allows the graph to be printed and options relevant to the plot area can be altered. Firstly, I will talk about the main Form which I have created.

The user has to choose a function by selecting one of nine buttons that are arranged in a keypad style. Once a function is selected, a graph of the original function with a highlighted point (with an initialised X-coordinate) is plot onto the PlotPanel component for the user to view. Simultaneously, a second graph with initialised transformations of the selected function and a highlighted point (of the same x-coordinate of the original graph) is plot on top of the original graph.

The user can then transform the second graph using their input. This is done using four sliders coupled with text boxes, and three tick boxes (giving a total of 7 function transformations). A transformation that can have its magnitude varied is represented by a slider coupled with textbox containing the position of the slider. As the slider is varied, the function is transformed and the graph is redrawn on the plot area with the corresponding highlighted point. Simultaneously, the text box under the slider updates and displays the integer value of the transformation magnitude. These text boxes can actually be used as a second way to input the magnitude of the function transformation and vary it. An integer value can be directly inserted into the textbox and the graph and slider will update accordingly, or up or down arrow attached to the box can pressed to increase/decrease magnitude (graph updates accordingly). One or more of these types of functions can be applied simultaneously to the same function, allowing compound transformations.

Tick boxes are used to represent non-magnitude transformations. When a tick box is checked the transformations is applied and the graph is updated and vice versa for when it is unchecked. This is what happens for the $[-f(x)]$ and $[f(-x)]$ transformations but for the $[\frac{1}{f(x)}]$ transformation, it is slightly different. The original and transformed graphs are hidden and the inverse of the selected function in calculated and plot. I have made it so that $[\frac{1}{f(x)}]$ transformation is a stand-alone one, meaning can't be applied with other transformations. The user can also vary the X-coordinates of the highlighted points on the graphs. This is done using the same slider and textbox system explained earlier. As

As the graph area is transformed and updated, dynamic information about the transformation is also displayed in a text area above the plots. This is along with the Cartesian coordinates of the two highlighted points on the original and transformed plots (which are updated as the points change).

In addition there are buttons which allow the transformed graph to be reset, the printing of the graph and the variation of certain graph options. When the reset button is clicked, all the sliders, textboxes and tick boxes are set back to their original states/positions before user variation. The pressing of the option displays a new form (which I have designed) called options box, which has settings for the graph and page setup for the printer. The graph settings allow the user to turn on/off the X and Y labels and graph grid. In addition, the user can also vary the graph area background colour within the options box. In terms of the page setup, the user can vary the orientation, margins and type of the print page (this is a standard Microsoft windows GUI dialog). Finally, the print button itself displays a (standard Microsoft windows) print dialog which allows the printer and number of copies to be set, and execution of the print job.

2. ACKNOWLEDGEMENT OF NON-CANDIDATE CODING

Here I am going to formally acknowledge the use of any components or units of coding which I have not designed.

First of all, as I have already mentioned earlier, I am using a Lazarus user-generated component called PlotPanel. I had to download and install this as a package to the Lazarus IDE. In order to use this component, I have to drag it from the IDE component palette into a Form in design mode. Then, I have to create a variable of this component and I can get access to all its fields and methods (which I haven't coded). However, I still have to manipulate the component through coding so it plots and transforms each individual point of the function I require.

Secondly, the IDE has an auto-complete feature which creates method stubs and class definitions for Forms that are being used. I have only used it twice to create class definitions for **TForm1** and **TOptionsbox** and stubs of their methods. I have still had to code the implementation sections of both the classes and write the code for each individual method that I use.

3. DETAILED ALGORITHM DESIGN

For this section I am going to show the algorithm on the right and beside it show the relevant coding from my actual program.

SET XY AXES LIMITS

Var Xmin = minimum X-Axis Value; {Specified by programmer}
Var Xmax = maximum X-Axis Value; {Specified by programmer}
Var Ymin = minimum Y-Axis Value; {Specified by programmer}
Var Ymax = maximum Y-Axis Value; {Specified by programmer}
 	
Set PlotPanel X-Axis range from **Xmin** to **Xmax;**
Set PlotPanel Y-Axis range from **Ymin** to **Ymax;**
	Procedure TFunction.SetXYAxesLimits(minXval,maxXval, minYval,maxYval: integer; PlotPanel1: TPlotPanel);
	Begin
	Xmin:= minXval;
	Xmax:= maxXval;
	Ymin:= minYval;
	Ymax:= maxYval;
	PlotPanel1.Freeze(True);
	PlotPanel1.XMin:= Xmin;
	PlotPanel1.XMax:= Xmax;
	PlotPanel1.YMin:= Ymin;
	PlotPanel1.YMax:= Ymax;
	end;

SET XY AXES INTERVALS AND LAYERS

Var Xinter = X-Axis interval value; {Specified by programmer}
Var Yinter = Y-Axis interval value; {Specified by programmer}
 	
Set PlotPanel X Axis intervals equal to **Xinter;**
Set PlotPanel Y Axis intervals equal to **Yinter;**
 	
Create PlotPanel *layer* for X-Axis;
Create PlotPanel *layer* for Y-Axis;
Create PlotPanel *layer* for Original graph;
Create PlotPanel *layer* for Transformed graph;
Create PlotPanel *layer* for Inverse graph;
Create PlotPanel *layer* for highlighted point on Original graph;
Create PlotPanel *layer* for highlighted point on Transformed graph;
	Procedure TFunction.SetXYAxesIntervalsAndLayers(Xinter,Yinter: integer; PlotPanel1: TPlotPanel);
	
	Begin
	PlotPanel1.XInterval:= Xinter;
	PlotPanel1.YInterval:= Yinter;
	PlotPanel1.LayerOptions(0,pmLine,3);
	PlotPanel1.LayerOptions(1,pmLine,2);
	PlotPanel1.LayerOptions(2,pmLine,2);
	PlotPanel1.LayerOptions(3,pmLine,2);
	PlotPanel1.LayerOptions(4,pmLine,2);
	PlotPanel1.LayerOptions(5,pmdot,8);
	PlotPanel1.LayerOptions(6,pmdot,8);
	PlotPanel1.ClearData;
	End;

PLOT XY AXES

Var Xmin = minimum X-Axis Value; {Specified by programmer}
Var Xmax = maximum X-Axis Value; {Specified by programmer}
Var Ymin = minimum Y-Axis Value; {Specified by programmer}
Var Ymax = maximum Y-Axis Value ; {Specified by programmer}
Var count = 0;
 	
For count ← **Xmin** to **Xmax**
{
Plot point on PlotPanel with coordinates of (**count**,0);
}
{This produces a straight line which acts as the X-Axis}
 	
For count ← **Ymin** to **Ymax**
{
Plot point on PlotPanel with coordinates of (0,**count**);
}
{This produces a straight line which acts as the Y-Axis}
	Procedure TFunction.PlotXYAxes
	(PlotPanel1: TPlotPanel);
	Var
	i : integer;
	Begin
	for i:= Xmin to Xmax
	do PlotPanel1.AddXY(i,0,clblack,1);
	for i:= Ymin to Ymax
	do PlotPanel1.AddXY(0,i,clblack,2);
	end;

GET FUNCTION TRANSFORMATION MAGNITUDES

Var a = Magnitude of Y-Translation transformation;
Var b = Magnitude of X-Translation transformation;
Var c = Magnitude of Y-Stretch transformation;
Var d = Magnitude of X-Stretch transformation;
Var e = Magnitude of X-Coordinate;
Var f = Y-Axis Reflection;
Var g =X-Axis Reflection;
Var InvX = Inverse Transformation;
 	
Get values of **a,b,c,d** and **e** from graph transformation user interface form
 	
If (**f** is selected in graph transformation user interface form)
f ← -1;
Else
f ← 1;
 	
If (**g** is selected graph transformation user interface form)
g ← -1;
Else
g ← 1;
 	
If (**InvX** is selected graph transformation user interface form)
InvX ← true;
Else
InvX ← false;
	Procedure
	TFunction.GetFunctionTransformationMagnitudes(
	Form1: TForm1);
	Begin
	a:= Form1.YTranslation.Position;
	b:= Form1.XTranslation.Position;
	c:= Form1.Ystretch.Position;
	d:= Form1.XStretch.Position;
	e:= Form1.XCoordinate.Position;
	if Form1.ReflectX.State = cbChecked
	then f := -1
	else f := 1;
	if Form1.ReflectY.State = cbChecked
	then g := -1
	else g := 1;
	if Form1.InverseX.State = cbChecked
	then InvX := true
	else InvX := false;
	end;

PLOT FUNCTION

```
Var  count = 0;
Var YValue = y-value to be plot;
Var countmin = The minimum value of the count;
Var countmax = The maximum value of the count;

For count ← countmin to countmax
{
  YValue  ← Transform count using mathematical expression corresponding to function;
  Plot point on PlotPanel with coordinates (count, YValue) ;
}
```

```
//The following are all the same method. Each procedure is just a different implementation of it for each subclass of TFunction. There are nine different implementations for each of the nine functions.
Procedure TLinear.PlotFunction(PlotPanel1: TPlotPanel);
Var
 i : integer;
Begin
  for i:= -10 to +10 do PlotPanel1.AddXY(i, i,clred,0);
end;

Procedure TQuadratic.PlotFunction(PlotPanel1:
TPlotPanel);
Var
 i : integer;
Begin
  for i:= -1000 to +1000 do
    Begin
      YValue := (0.01*i)*(0.01*i);
      PlotPanel1.AddXY((0.01*i),YValue ,clred,0);
    end;
end;

Procedure TCubic.PlotFunction(PlotPanel1: TPlotPanel);
Var
 i : integer;
Begin
  for i:= -1000 to +1000 do
    Begin
      YValue := (0.01*i)*(0.01*i)*(0.01*i);
      PlotPanel1.AddXY((0.01*i),YValue ,clred,0);
    end;
end;

Procedure TSine.PlotFunction(PlotPanel1: TPlotPanel);
Var
 i : integer;
Begin
  for i:=-360 to 360 do
    begin
      YValue:=sin((PI/180)*i);
      PlotPanel1.AddXY(i,YValue,clred,0);
    end;
end;

Procedure TCosine.PlotFunction(PlotPanel1: TPlotPanel);
Var
 i : integer;
Begin
 for i:=-360 to 360 do
    begin
      YValue:=cos((PI/180)*i);
```

```
      PlotPanel1.AddXY(i,YValue,clRed,0);
    end;
end;

Procedure TTangent.PlotFunction(PlotPanel1:
TPlotPanel);
Var
 i : integer;
Begin
for i:=-360 to 360 do
   begin
   if i mod 90 = 0
     then
      begin
       YValue:=(tan((PI/180)*(i-0.001)));
       PlotPanel1.AddXY(i,YValue,clred,0);
      end
     else
      begin
       YValue:=(tan((PI/180)*i));
       PlotPanel1.AddXY(i,YValue,clRed,0);
      end;
   end;
end;

Procedure TExponential.PlotFunction(PlotPanel1:
TPlotPanel);
Var
 i : integer;
Begin
  for i:= -1000 to +300 do
    Begin
     YValue := Exp(0.01*i);
     PlotPanel1.AddXY((0.01*i),YValue ,clred,0);
    end;
end;

Procedure TInverse.PlotFunction(PlotPanel1:
TPlotPanel);
Var
 i : integer;
Begin
  for i:= -1000 to -1 do
    Begin
     YValue := (1/(0.01*i));
     PlotPanel1.AddXY((0.01*i),YValue ,clred,0);
    end;
  for i:= 1 to 1000  do
    Begin
     YValue := (1/(0.01*i));
     PlotPanel1.AddXY((0.01*i),YValue ,clred,0);
    end;
end;

Procedure TModulus.PlotFunction(PlotPanel1:
TPlotPanel);
Var
```

	i : integer;
	Begin
	for i:= -1000 to 1000 do
	Begin
	YValue := (abs(0.01*i)); //"abs(x)" returns the absolute
	PlotPanel1.AddXY((0.01*i),YValue ,clred,0); // value of x
	end;
	end;

APPLY TRANSFORMATION

Var a = Magnitude of Y-Translation transformation;
Var b = Magnitude of X-Translation transformation;
Var c = Magnitude of Y-Stretch transformation;
Var d = Magnitude of X-Stretch transformation;
Var f = Y-Axis Reflection;
Var g =X-Axis Reflection;
Var InvX = Inverse Transformation
Var = count = 0;
Var YValue = y-value to plot;
Var countmin = The minimum value of the count;
Var countmax = The maximum value of the count;

For count ← **countmin** to **countmax**
{
YValue ← **Transform *count*** using transformed mathematical expression of function involving **a,b,c,d,f** and **g;**
Plot point on PlotPanel with coordinates **(count, YValue);**
}

If *InvX* <-- true
{
Hide all *Layers* currently visible except X and Y Axes;
For count ← **countmin** to **countmax**
{
YValue ← **Transform *count*** using inverse mathematical expression of function;
Plot point of PlotPanel with coordinates **(count, YValue)** on *Layer* created for the inverse function;
}
Show *Layer* with plot of inverse function;
}

```
//The following are all the same method. Each procedure is just a different implementation of it for each subclass of TFunction. There are nine different implementations for each of the nine functions.


Procedure TLinear.ApplyTransformation(PlotPanel1:
TPlotPanel);
Var
 i : integer;
Begin
  PlotPanel1.Freeze(True);
  PlotPanel1.HideLayer(3);
  for i:= -10 to +10 do
   Begin
    YValue:=f *g*((d*(c*(i+b)))+a);
    PlotPanel1.AddXY(i, YValue ,clblue,3);
   end;
  PlotPanel1.UnHideLayer(3);
  if InvX = true
   Then
    Begin
     PlotPanel1.HideLayer(0);
     PlotPanel1.HideLayer(3);
     PlotPanel1.HideLayer(4);
     for i:= -1000 to -1 do
      Begin
       YValue := (1/(0.01*i));
       PlotPanel1.AddXY((0.01*i),YValue ,clmaroon,4);
      end;
     for i:= 1 to 1000  do
      Begin
       YValue := (1/(0.01*i));
       PlotPanel1.AddXY((0.01*i),YValue ,clmaroon,4);
      end;
     PlotPanel1.UnHideLayer(4);
    end;
end;


Procedure TQuadratic.ApplyTransformation(PlotPanel1:
TPlotPanel);
Var
 i : integer;
Begin
  PlotPanel1.Freeze(True);
  PlotPanel1.HideLayer(3);
  for i:= -1000 to +1000 do
   Begin
    YValue := f
```

```
*(c*((g*(d*((0.01*i)+b)))*(g*(d*((0.01*i)+b))))+a);
      PlotPanel1.AddXY((0.01*i),YValue ,clblue,3);
     end;
   PlotPanel1.UnHideLayer(3);
   if InvX = true
    Then
     Begin
      PlotPanel1.HideLayer(0);
      PlotPanel1.HideLayer(3);
      PlotPanel1.HideLayer(4);
      for i:= -1000 to -1 do
       Begin
        YValue := 1/((0.01*i)*(0.01*i));
        PlotPanel1.AddXY((0.01*i),YValue ,clmaroon,4);
       end;
      for i:= 1 to 1000  do
       Begin
        YValue := 1/((0.01*i)*(0.01*i));
        PlotPanel1.AddXY((0.01*i),YValue ,clmaroon,4);
       end;
      PlotPanel1.UnHideLayer(4);
     end;
end;

Procedure TCubic.ApplyTransformation(PlotPanel1: TPlotPanel);
Var
 i : integer;
Begin
  PlotPanel1.Freeze(True);
  PlotPanel1.HideLayer(3);
  for i:= -1000 to +1000 do
   Begin
    YValue := f
*(c*((g*(d*((0.01*i)+b)))*(g*(d*((0.01*i)+b)))*(g*(d*((0.01*i)+b
))))+a);
     PlotPanel1.AddXY((0.01*i),YValue ,clblue,3);
   end;
  PlotPanel1.UnHideLayer(3);
  if InvX = true
   Then
    Begin
     PlotPanel1.HideLayer(0);
     PlotPanel1.HideLayer(3);
     PlotPanel1.HideLayer(4);
     for i:= -1000 to -1 do
      Begin
       YValue := 1/((0.01*i)*(0.01*i)*(0.01*i));
       PlotPanel1.AddXY((0.01*i),YValue ,clmaroon,4);
      end;
     for i:= 1 to 1000  do
      Begin
       YValue := 1/((0.01*i)*(0.01*i)*(0.01*i));
       PlotPanel1.AddXY((0.01*i),YValue ,clmaroon,4);
      end;
     PlotPanel1.UnHideLayer(4);
    end;
end;
```

```
Procedure TSine.ApplyTransformation(PlotPanel1: TPlotPanel);
Var
 i : integer;
Begin
  PlotPanel1.Freeze(True);
  PlotPanel1.HideLayer(3);
  for i:=-360 to 360 do
   begin
    YValue:=f *(c*(sin(g*(d*(((PI/180)*i)+b))))+a);
    PlotPanel1.AddXY(i,YValue,clblue,3);
   end;
  PlotPanel1.UnHideLayer(3);
  if InvX = true
   Then
    Begin
     PlotPanel1.HideLayer(0);
     PlotPanel1.HideLayer(3);
     PlotPanel1.HideLayer(4);
     for i:=-360 to 360 do
      begin
       if i mod 180 = 0
        then
         begin
          YValue:=(csc((PI/180)*(i-0.001)));    //csc(x)= 1/sin(x)
          PlotPanel1.AddXY(i,YValue,clmaroon,4);
         end
        else
         begin
          YValue:=(csc((PI/180)*i));
          PlotPanel1.AddXY(i,YValue,clmaroon,4);
         end;
      end;
     PlotPanel1.UnHideLayer(4);
    end;
end;

Procedure TCosine.ApplyTransformation(PlotPanel1:
TPlotPanel);
Var
 i : integer;
Begin
  PlotPanel1.Freeze(True);
  PlotPanel1.HideLayer(3);
  for i:=-360 to 360 do
   begin
    YValue:=f *(c*(cos(g*(d*(((PI/180)*i)+b))))+a);
    PlotPanel1.AddXY(i,YValue,clblue,3);
   end;
  PlotPanel1.UnHideLayer(3);
  if InvX = true
   Then
    Begin
     PlotPanel1.HideLayer(0);
     PlotPanel1.HideLayer(3);
     PlotPanel1.HideLayer(4);
     for i:= -360 to 360 do
```

```
        begin
         if (i=-270) or (i=-90) or (i=90) or (i=270)
          then
           begin
            YValue:=(sec((PI/180)*(i-0.001)));    //sec(x)=1/cos(x)
            PlotPanel1.AddXY(i,YValue,clmaroon,4);
           end
          else
           begin
            YValue:=(sec((PI/180)*i));
            PlotPanel1.AddXY(i,YValue,clmaroon,4);
           end;
        end;
       PlotPanel1.UnHideLayer(4);
     end;
end;

Procedure TTangent.ApplyTransformation(PlotPanel1:
TPlotPanel);
Var
 i : integer;
Begin
  PlotPanel1.Freeze(True);
  PlotPanel1.HideLayer(3);
  for i:=-360 to 360 do
   begin
   if i mod 90 = 0
     then
      begin
       YValue:=f *(c*(tan(g*(d*(((PI/180)*(i-0.001))+b))))+a);
//got a problem with "d"
       PlotPanel1.AddXY(i,YValue,clblue,3);
      end
     else
      begin
       YValue:=f *(c*(tan(g*(d*(((PI/180)*i)+b))))+a);
       PlotPanel1.AddXY(i,YValue,clblue,3);
      end;
  end;
 PlotPanel1.UnHideLayer(3);
 if InvX = true
  Then
   Begin
    PlotPanel1.HideLayer(0);
    PlotPanel1.HideLayer(3);
    PlotPanel1.HideLayer(4);
    for i:= -360 to 360 do
     begin
      if i mod 180 = 0
       then
        begin
         YValue:=(cot((PI/180)*(i-0.001)));    //cot(x)=1/tan(x)
         PlotPanel1.AddXY(i,YValue,clmaroon,4);
        end
       else
        begin
         YValue:=(cot((PI/180)*i));
```

```
            PlotPanel1.AddXY(i,YValue,clmaroon,4);
           end;
        end;
       PlotPanel1.UnHideLayer(4);
      end;
end;

Procedure TExponential.ApplyTransformation(PlotPanel1:
TPlotPanel);
Var
 i : integer;
Begin
  PlotPanel1.Freeze(True);
  PlotPanel1.HideLayer(3);
  for i:= -1000 to +1000 do
   Begin
    YValue :=f *(c*(Exp(g*(d*(0.01*i)+b)))+a);          //Problem
with "d" and "b" extremes
    PlotPanel1.AddXY((0.01*i),YValue ,clblue,3);   //try
decreasing values maybe?
   end;
  PlotPanel1.UnHideLayer(3);
  if InvX = true
   Then
    Begin
     PlotPanel1.HideLayer(0);
     PlotPanel1.HideLayer(3);
     PlotPanel1.HideLayer(4);
     for i:= -1000 to 1000 do
      Begin
       YValue := 1/(exp(0.01*i));
       PlotPanel1.AddXY((0.01*i),YValue ,clMaroon,4);
      end;
     PlotPanel1.UnHideLayer(4);
    end;
end;

Procedure TInverse.ApplyTransformation(PlotPanel1:
TPlotPanel);
Var
 d1 : extended;
 i : integer;
Begin
  PlotPanel1.Freeze(True);
  PlotPanel1.HideLayer(3);
  d1:= d;
  if d1 = 0
   then d1:= d1 + 0.01;   //error validation
  for i:= -1000 to (-1-(100*b)) do
   Begin
    YValue := f *(c*(1/(g*(d1*((0.01*i)+b))))+a);          //"b"
due to changing assymptote (fixed)
    PlotPanel1.AddXY((0.01*i),YValue ,clblue,3);   //"d" due to
zero
   end;
  for i:= (1-(100*b)) to 1000  do
   Begin
```

```
      YValue := f *(c*(1/(g*(d1*((0.01*i)+b))))+a);
      PlotPanel1.AddXY((0.01*i),YValue ,clblue,3);
     end;
   PlotPanel1.UnHideLayer(3);
   if InvX = true
    Then
     Begin
      PlotPanel1.HideLayer(0);
      PlotPanel1.HideLayer(3);
      PlotPanel1.HideLayer(4);
      for i:= -1000 to 1000 do
       Begin
        YValue :=(0.01*i);
        PlotPanel1.AddXY((0.01*i),YValue ,clmaroon,4);
       end;
      PlotPanel1.UnHideLayer(4);
     end;
end;

Procedure TModulus.ApplyTransformation(PlotPanel1:
TPlotPanel);
Var
 i : integer;
Begin
  PlotPanel1.Freeze(True);
  PlotPanel1.HideLayer(3);
  for i:= -1000 to 1000 do
   Begin
    YValue := f *(c*(Abs(g*(d*(0.01*i)+b)))+a);
    PlotPanel1.AddXY((0.01*i),YValue ,clblue,3);
   end;
  PlotPanel1.UnHideLayer(3);
  if InvX = true
   Then
    Begin
     PlotPanel1.HideLayer(0);
     PlotPanel1.HideLayer(3);
     PlotPanel1.HideLayer(4);
     for i:= -1000 to -1 do
      Begin
       YValue :=1/(Abs(0.01*i));
       PlotPanel1.AddXY((0.01*i),YValue ,clmaroon,4);
      end;
     for i:= 1 to 1000 do
      Begin
       YValue :=1/(Abs(0.01*i));
       PlotPanel1.AddXY((0.01*i),YValue ,clmaroon,4);
      end;
     PlotPanel1.UnHideLayer(4);
    end;
end;
```

Var a = Magnitude of Y-Translation transformation;
Var b = Magnitude of X-Translation transformation;
Var c = Magnitude of Y-Stretch transformation;

Var d = Magnitude of X-Stretch transformation;
Var e = Magnitude of X-Coordinate
Var f = Y-Axis Reflection;
Var g =X-Axis Reflection;
Var InvX = Inverse Transformation
Var YValue = y-value to plot;

If InvX = false
{
Hide layers created for highlighted points {so that we can plot on them without any flicker appearing on the graph}
YValue ← **Transform *count*** using transformed mathematical expression of function involving use of **a,b,c,d,e,f** and **g**
Plot point on PlotPanel with coordinates **(e, YValue)** on *layer* assigned for "highlighted point on transformed graph"
Plot point on PlotPanel with coordinates **(e,** (selected function applied to **e))** on *layer* assigned for "highlighted point on Original graph"
}

```
//HighlightPoints
//The following are all the same method. Each
procedure is just a different implementation of it for
each subclass of TFunction. There are nine different
implementations for each of the nine functions.


Procedure TLinear.HighlightPoints(PlotPanel1:
TPlotPanel);
Begin
 if InvX = false
  then
    Begin
     PlotPanel1.HideLayer(5);
     PlotPanel1.HideLayer(6);
     YValue:=f *g*((d*(c*(e+b)))+a);
     PlotPanel1.AddXY(e, YValue ,clblue,5);
     PlotPanel1.AddXY(e,e,clred,6);
     PlotPanel1.UnHideLayer(0);
     PlotPanel1.UnHideLayer(5);
     PlotPanel1.UnHideLayer(6);
     PlotPanel1.HideLayer(4);
    end;
  PlotPanel1.Freeze(false);
end;

Procedure TQuadratic.HighlightPoints(PlotPanel1:
TPlotPanel);
Begin
 if InvX = false
  then
    Begin
     PlotPanel1.HideLayer(5);
     PlotPanel1.HideLayer(6);
     YValue := f
*(c*((g*(d*((e)+b)))*(g*(d*((e)+b))))+a);
     PlotPanel1.AddXY(e, YValue ,clblue,5);
     PlotPanel1.AddXY(e,e*e,clred,6);
     PlotPanel1.UnHideLayer(0);
     PlotPanel1.UnHideLayer(5);
     PlotPanel1.UnHideLayer(6);
     PlotPanel1.HideLayer(4);
    end;
 PlotPanel1.Freeze(false);
end;

Procedure TCubic.HighlightPoints(PlotPanel1:
TPlotPanel);
Begin
 if InvX = false
  then
    Begin
```

```
        PlotPanel1.HideLayer(5);
        PlotPanel1.HideLayer(6);
        YValue := f
 *(c*((g*(d*((e)+b)))*(g*(d*((e)+b)))*(g*(d*((e)+b))))+a);
        PlotPanel1.AddXY(e, YValue ,clblue,5);
        PlotPanel1.AddXY(e,e*e*e,clred,6);
        PlotPanel1.UnHideLayer(0);
        PlotPanel1.UnHideLayer(5);
        PlotPanel1.UnHideLayer(6);
        PlotPanel1.HideLayer(4);
      end;
  PlotPanel1.Freeze(false);
end;

Procedure TSine.HighlightPoints(PlotPanel1:
TPlotPanel);
Begin
  if InvX = false
    then
      Begin
        PlotPanel1.HideLayer(5);
        PlotPanel1.HideLayer(6);
        YValue:= f
 *(c*(sin(g*(d*(((PI/180)*(e*72))+b))))+a);
        PlotPanel1.AddXY(e*72, YValue ,clblue,5);

PlotPanel1.AddXY(e*72,sin((PI/180)*(e*72)),clred,6);
        PlotPanel1.UnHideLayer(0);
        PlotPanel1.UnHideLayer(5);
        PlotPanel1.UnHideLayer(6);
        PlotPanel1.HideLayer(4);
      end;
  PlotPanel1.Freeze(false);
end;

Procedure TCosine.HighlightPoints(PlotPanel1:
TPlotPanel);
Begin
  if InvX = false
    then
      Begin
        PlotPanel1.HideLayer(5);
        PlotPanel1.HideLayer(6);
        YValue:= f
 *(c*(cos(g*(d*(((PI/180)*(e*72))+b))))+a);
        PlotPanel1.AddXY(e*72, YValue ,clblue,5);

PlotPanel1.AddXY(e*72,cos((PI/180)*(e*72)),clred,6);
        PlotPanel1.UnHideLayer(0);
        PlotPanel1.UnHideLayer(5);
        PlotPanel1.UnHideLayer(6);
        PlotPanel1.HideLayer(4);
      end;
  PlotPanel1.Freeze(false);
end;

Procedure TTangent.HighlightPoints(PlotPanel1:
```

```
TPlotPanel);
Begin
 if InvX = false
  then
    Begin
     PlotPanel1.HideLayer(5);
     PlotPanel1.HideLayer(6);
     YValue:= f *(c*(tan(g*(d*(((PI/180)*((e-
0.001)*72))+b))))+a);
     PlotPanel1.AddXY(e*72, YValue ,clblue,5);
     PlotPanel1.AddXY(e*72,tan((PI/180)*((e-
0.001)*72)),clred,6);
     PlotPanel1.UnHideLayer(0);
     PlotPanel1.UnHideLayer(5);
     PlotPanel1.UnHideLayer(6);
     PlotPanel1.HideLayer(4);
    end;
 PlotPanel1.Freeze(false);
end;

Procedure TExponential.HighlightPoints(PlotPanel1:
TPlotPanel);
Begin
 if InvX = false
  then
    Begin
     PlotPanel1.HideLayer(5);
     PlotPanel1.HideLayer(6);
     YValue:= f *(c*(Exp(g*(d*(e)+b)))+a);
     PlotPanel1.AddXY(e, YValue ,clblue,5);
     PlotPanel1.AddXY(e,Exp(e),clred,6);
     PlotPanel1.UnHideLayer(0);
     PlotPanel1.UnHideLayer(5);
     PlotPanel1.UnHideLayer(6);
     PlotPanel1.HideLayer(4);
    end;
 PlotPanel1.Freeze(false);
end;

Procedure TInverse.HighlightPoints(PlotPanel1:
TPlotPanel);
Var
 d1 : real;
Begin
 d1:= d;
 if d1 = 0
  then d1:= d1 + 0.01;
 if InvX = false
  then
    Begin
     PlotPanel1.HideLayer(5);
     PlotPanel1.HideLayer(6);
     YValue:= f *(c*(1/(g*(d1*((e-0.001)+b))))+a);
     PlotPanel1.AddXY(e, YValue ,clblue,5);
     PlotPanel1.AddXY(e,(1/(e-0.001)),clred,6);
     PlotPanel1.UnHideLayer(0);
     PlotPanel1.UnHideLayer(5);
```

```
        PlotPanel1.UnHideLayer(6);
        PlotPanel1.HideLayer(4);
      end;
  PlotPanel1.Freeze(false);
end;

Procedure TModulus.HighlightPoints(PlotPanel1:
TPlotPanel);
Begin
  if InvX = false
    then
      Begin
        PlotPanel1.HideLayer(5);
        PlotPanel1.HideLayer(6);
        YValue := f *(c*(Abs(g*(d*(e)+b)))+a);
        PlotPanel1.AddXY(e, YValue ,clblue,5);
        PlotPanel1.AddXY(e,Abs(e),clred,6);
        PlotPanel1.UnHideLayer(0);
        PlotPanel1.UnHideLayer(3);
        PlotPanel1.UnHideLayer(5);
        PlotPanel1.UnHideLayer(6);
        PlotPanel1.HideLayer(4);
      end;
  PlotPanel1.Freeze(false);
end;
```

OPTIONS BOX

Create Options box Form

If (Options Button on Main form clicked)
    Set Options box Form to visible;

If (Tickbox corresponding to XLabel is checked)
    Set XMarks on PlotPanel to true;
Else
    Set XMarks on PlotPanel to false;

If (Tickbox corresponding to YLabel is checked0
    Set YMarks on PlotPanel to true;
Else
    Set YMarks on PlotPanel to false;

If (Tickbox corresponding to GridLines is checked)
    Set Grid Color on PlotPanel to Black;
Else
    PlotPanel Grid Colour ← PlotPanel Back Colour;

If (BackColour colour selector changes)
    PlotPanel Back Colour ← Colour currently selected;

If (Page setup clicked)
    Execute PageSetup dialog;

```
procedure TForm1.OptionsButtonClick(Sender:
TObject);
begin
  Optionsbox.Visible:= true;
  Optionsbox.FormStyle:= fsStayOnTop;
end;

// OptionsBox unit. I had to create a separate unit for
//this form.

unit Unit2;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils, FileUtil, LResources, Forms,
Controls, Graphics, Dialogs,
  ButtonPanel, ExtCtrls, StdCtrls, PrintersDlgs;

type

  { TOptionsBox }

  TOptionsBox = class(TForm)
    PageSetup: TButton;
    CloseButn: TButton;
    XLabel: TCheckBox;
    YLabel: TCheckBox;
    GridLines: TCheckBox;
    GraphColour: TColorButton;
    PageSetupDialog: TPageSetupDialog;
    GraphSettings: TPanel;
    PrinterSettings: TPanel;
    Graph: TStaticText;
    Printer: TStaticText;
    procedure GraphColourColorChanged(Sender:
TObject);
    procedure GridLinesClick(Sender: TObject);
    procedure CloseButnClick(Sender: TObject);
    procedure PageSetupClick(Sender: TObject);
    procedure XLabelClick(Sender: TObject);
    procedure YLabelClick(Sender: TObject);
  private
    { private declarations }
  public
    { public declarations }
  end;

var
  OptionsBox: TOptionsBox;
```

```
implementation

uses Unit1;

{ TOptionsBox }


procedure TOptionsBox.XLabelClick(Sender: TObject);
begin
  if XLabel.State = cbChecked
    then Form1.PlotPanel1.XMarks:= true
    else Form1.PlotPanel1.XMarks:= false;
end;

procedure TOptionsBox.YLabelClick(Sender: TObject);
begin
  if YLabel.State = cbChecked
    then Form1.PlotPanel1.YMarks:= true
    else Form1.PlotPanel1.YMarks:= false;
end;

procedure TOptionsBox.PageSetupClick(Sender:
TObject);
begin
  PageSetupDialog.Execute = true;
end;

procedure TOptionsBox.GridLinesClick(Sender:
TObject);
begin
  if GridLines.State = cbChecked
    then Form1.PlotPanel1.GridColor:= clDefault
    else Form1.PlotPanel1.GridColor:=
Form1.PlotPanel1.BackColor;

end;

procedure TOptionsBox.CloseButnClick(Sender:
TObject);
begin
  OptionsBox.Close;
end;

procedure
TOptionsBox.GraphColourColorChanged(Sender:
TObject);
begin
  Form1.PlotPanel1.BackColor:=
GraphColour.ButtonColor;
end;

initialization
  {$I unit2.lrs}

end.
```

RESET GRAPH

If (Reset Button clicked on Main Form)
{
Intialise slider positions
Intialise text boxes
Intialise Check boxes
Update PlotPanel and TextArea containing graph information
}
	procedure TForm1.ResetButtonClick(Sender: TObject);
	begin
	YTranslation.Position:= 1;
	XTranslation.Position:= 0;
	XStretch.Position:= 1;
	YStretch.Position:= 1;
	XCoordinate.Position:= 1;
	YTransEdit.Value:= 1;
	XTransEdit.Value:= 0;
	YStretchEdit.Value:= 1;
	XStretchEdit.Value:= 1;
	XCoordEdit.Value:= 1;
	ReflectX.State:= cbUnchecked;
	ReflectY.State:= cbUnchecked;
	InverseX.State:= cbUnchecked;
	UpdateTransformedGraph;
	end;

PRINT GRAPH

If (Print Button clicked on Main Form)
{
Create PrinterDialog;
 	
If (PrinterDialog executed)
{
No. Of Copies ← Copy Number specified in Printer Dialog
Print Bitmap image of PlotPanel Graph Area
}
	procedure TForm1.PrintButtonClick(Sender: TObject);
	
	const
	TOTAL_PAGES = 1; // How many pages to print
	var
	printDialog : TPrintDialog;
	page, startPage, endPage : Integer;
	MarginX, MarginY: longint;
	
	begin
	// Create a printer selection dialog
	printDialog := TPrintDialog.Create(Form1);
	
	// Set up print dialog options
	printDialog.MinPage := 1; // First allowed page number
	printDialog.MaxPage := TOTAL_PAGES; // Highest allowed page number
	printDialog.ToPage := TOTAL_PAGES; // 1 to ToPage page range allowed
	printDialog.Options := [poPageNums]; // Allow page range selection
	
	// if the user has selected a printer (or default), then print!

```
 if printDialog.Execute then
   begin

    // Set the printjob title - as it it appears in the print job
manager
    Printer.Title := 'Transformations of the graphs of functions';

    // Set the number of copies to print each page
    // This is crude - it doies not take Collation into account
    Printer.Copies := printDialog.Copies;

    //Set margins

    MarginX :=  optionsbox.PageSetupDialog.Margins.left;
    MarginY :=  optionsbox.PageSetupDialog.Margins.top;


    // Start printing
    Printer.BeginDoc;

    // Has the user selected a page range?
    if printDialog.PrintRange = prPageNums then
     begin
      startPage := printDialog.FromPage;
      endPage   := printDialog.ToPage;
     end
    else // All pages
     begin
      startPage := 1;
      endPage   := TOTAL_PAGES;
     end;

    // Set up the start page number
    page := startPage;

    // Show a message saying we are starting a page
    ShowMessagePos('Starting to print page
'+IntToStr(page),300,300);

    // Allow Windows to keep processing messages
    Application.ProcessMessages;

Printer.Canvas.Rectangle(optionsbox.PageSetupDialog.Margins);
    Printer.Canvas.Draw(MarginX, MarginY,
plotpanel1.PlotBMP);

    // Finish printing
    Printer.EndDoc;
  end;
end;
```

4. PROCEDURE/METHOD AND VARIABLE LISTS

PROCEDURES/METHODS

NON-CLASS PROCEDURES

Procedure UpdateTransformedGraph;
//This combines certain methods out of TFunction so that the Transformed graph is fully update according to the positions of sliders and states of tickboxes. It also updates the text above the graph at the same time.

TFORM1 METHODS

This class has been produced so that I can create and manipulate a Form for my GUI. I coded the implementation for all of these procedures. The following are the class methods, not object methods. (I still have created and used objects of this class)

Procedure TForm1.PrintButtonClick(Sender: TObject);
//This is used to respond to the event when the Print button within my form is clicked. The Print dialog is made visible
Procedure TForm1.ReflectXClick(Sender: TObject);
//This is used to respond to the event when the tickbox corresponding to the "Reflection in X-Axis" transformation is clicked.
Procedure TForm1.ReflectYClick(Sender: TObject);
Procedure TForm1.InverseXClick(Sender: TObject);
Procedure TForm1.ResetButtonClick(Sender: TObject);
Procedure TForm1.OptionsButtonClick(Sender: TObject);
Procedure TForm1.CubicFunctionClick(Sender: TObject);
Procedure TForm1.ExponentialFunctionClick(Sender: TObject);
Procedure TForm1.InverseFunctionClick(Sender: TObject);
Procedure TForm1.LinearFunctionClick(Sender: TObject);
Procedure TForm1.ModulusFunctionClick(Sender: TObject);
Procedure TForm1.QuadraticFunctionClick(Sender: TObject);
Procedure TForm1.SineFunctionClick(Sender: TObject);
Procedure TForm1.CosineFunctionClick(Sender: TObject);
Procedure TForm1.SpinEdit1Change(Sender: TObject);
Procedure TForm1.SpinEdit2Change(Sender: TObject);
Procedure TForm1.SpinEdit3Change(Sender: TObject);
Procedure TForm1.SpinEdit4Change(Sender: TObject);
Procedure TForm1.SpinEdit5Change(Sender: TObject);
Procedure TForm1.TangentFunctionClick(Sender: TObject);
Procedure TForm1.XCoordinateChange(Sender: TObject);

```
Procedure TForm1.XStretchChange(Sender: TObject);
Procedure TForm1.XTranslationChange(Sender: TObject);
Procedure TForm1.YstretchChange(Sender: TObject);
Procedure TForm1.YTranslationChange(Sender: TObject);
```

TFUNCTION METHODS

There might be some variation here in comparison to the class definition of TFunction I did in the design section. The following are the class methods, not object methods. (I still have created and used objects of this class)

```
Procedure TFunction.SetXYAxesLimits (minXval,maxXval,minYval,maxYval: integer;
PlotPanel1: TPlotPanel);
Procedure TFunction.SetXYAxesIntervalsAndLayers (Xinter,Yinter: integer; PlotPanel1:
TPlotPanel);
Procedure TFunction.PlotXYAxes (PlotPanel1: TPlotPanel);
Procedure TFunction.GetFunctionTransformationMagnitudes (Form1: TForm1);
Procedure TFunction.CalculateTransformedGraphInfo;
Procedure TFunction.DisplayCartCoordAndTransFunctTxt (Form1: TForm1);
Procedure TFunction.ToggleboxFix (Form1: TForm1);
Procedure TFunction.PlotFunction (PlotPanel1: TPlotPanel);
Procedure TFunction.HighlightPoints (PlotPane1: TPlotPanel);
Procedure TFunction.ApplyTransformation (PlotPanel1: TPlotPanel);
Procedure TFunction.DisplayOrigTxtAndCalcCartCoord (Form1: TForm1);
```

TLINEAR METHODS

The following are the class methods, not object methods. (I still have created and used objects of this class)

```
Procedure TLinear.ToggleboxFix(Form1: TForm1);
Procedure TLinear.PlotFunction (PlotPanel1: TPlotPanel);
Procedure TLinear.ApplyTransformation(PlotPanel1: TPlotPanel);
Procedure TLinear.HighlightPoints(PlotPanel1: TPlotPanel);
Procedure TLinear.DisplayOrigTxtAndCalcCartCoord(Form1: TForm1);
```

These procedures are repeated eight more times for each of the other subclasses of TFuntion (for TQuadratic, TCubic, TSine, TCosine, TTangent, TExponential, TInverse and TModulus).

TOPTIONSBOX METHODS

The following are the class methods, not object methods. (I still have created and used objects of this class)

```
Procedure TOptionsBox.GraphColourColorChanged(Sender: TObject);
Procedure TOptionsBox.GridLinesClick(Sender: TObject);
Procedure TOptionsBox.CloseButnClick(Sender: TObject);
Procedure TOptionsBox.PageSetupClick(Sender: TObject);
Procedure TOptionsBox.XLabelClick(Sender: TObject);
Procedure TOptionsBox.YLabelClick(Sender: TObject);
```

VARIABLE/FIELD LISTS

TFUNCTION FIELDS

```
Xmin : integer;
Xmax : integer;
Ymin : integer;
Ymax : integer;
a : integer;
b : integer;
c : integer;
d : integer;
e : integer;
f  : integer;
g : integer;
InvX : boolean;
YValue : Extended;
Sa, Sb, Sc, Sd  : string;
TextInfo : String;
XRed, XBlue, YRed, YBlue  : string;
```

These fields are also inherited and used by all the subclasses of TFunction.

TFORM1 FIELDS

```
PrintButton: TButton;
ResetButton: TButton;
OptionsButton: TButton;
ReflectX: TCheckBox;
ReflectY: TCheckBox;
InverseX: TCheckBox;
FunctionsKeyPad: TPanel;
ButtonPanel: TPanel;
ToolPanel: TPanel;
TextPanel: TPanel;
FunctTransPanel: TPanel;
XCoordPanel: TPanel;
YTransEdit: TSpinEdit;
```

```
XTransEdit: TSpinEdit;
YStretchEdit: TSpinEdit;
XStretchEdit: TSpinEdit;
XCoordEdit: TSpinEdit;
YTransTxt: TStaticText;
XCoordLabel: TStaticText;
FunctTransLabel: TStaticText;
TransFunctTxt: TStaticText;
OrigFunctTxt: TStaticText;
OrigCartCoord: TStaticText;
TransCartCoord: TStaticText;
XTransTxt: TStaticText;
YStretchTxt: TStaticText;
XStretchTxt: TStaticText;
InverseFunction: TToggleBox;
ModulusFunction: TToggleBox;
PlotPanel1: TPlotPanel;
LinearFunction: TToggleBox;
SineFunction: TToggleBox;
CosineFunction: TToggleBox;
TangentFunction: TToggleBox;
ExponentialFunction: TToggleBox;
QuadraticFunction: TToggleBox;
CubicFunction: TToggleBox;
Ystretch: TTrackBar;
XStretch: TTrackBar;
XCoordinate: TTrackBar;
YTranslation: TTrackBar;
XTranslation: TTrackBar;
```

TOPTIONSBOX FIELDS

```
PageSetup: TButton;
CloseButn: TButton;
XLabel: TCheckBox;
YLabel: TCheckBox;
GridLines: TCheckBox;
GraphColour: TColorButton;
PageSetupDialog: TPageSetupDialog;
GraphSettings: TPanel;
PrinterSettings: TPanel;
Graph: TStaticText;
Printer: TStaticText;
```

UNIT 1 GLOBAL/LOCAL VARABLES

```
Form1: TForm1;          //Global
Function1: TFunction;  //Global
 i : integer; //Local variable used numerous times within different procedures as a counter
d1 : extended; //Local variable used within TInverse.ApplyTransformation
```

UNIT 2 GLOBAL/LOCAL VARIABLES

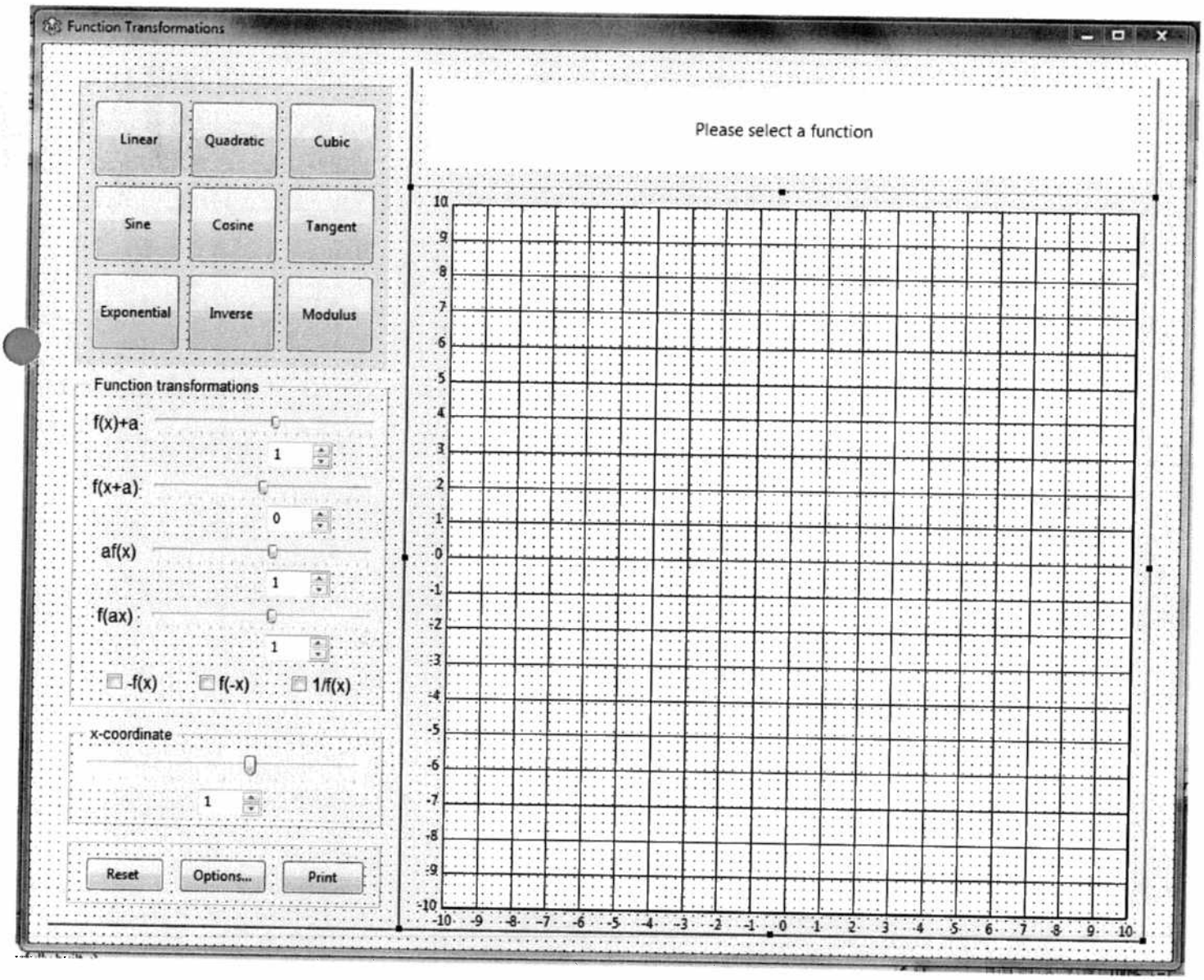
```
OptionsBox: TOptionsBox;  //Global
```

5. ANNOTATED CODE LISTING

Please see the "Code listing" section in the appendix

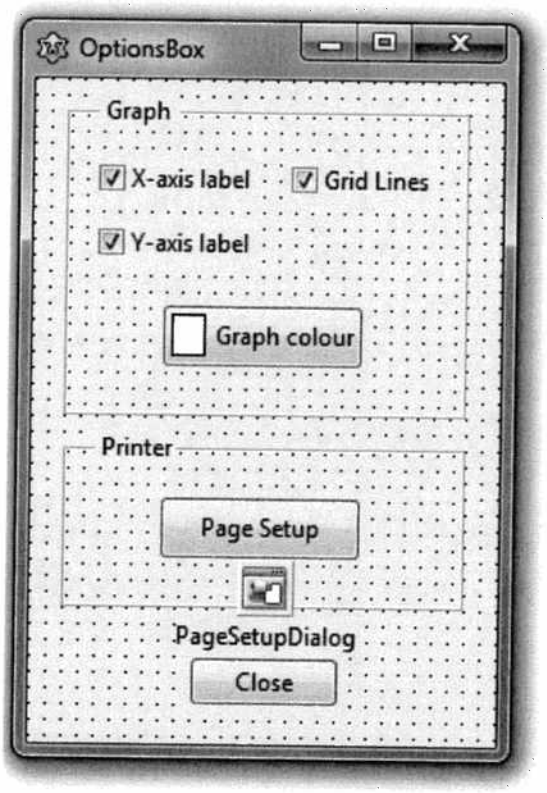
6. ANNOTATED DESIGN VIEWS OF SELF CREATED FORMS

FORM1 (MAIN FORM)



This nearly exactly the same as the Main form in the design section. Since, it is already explained there, I don't see the need to document it again.

OPTIONSBOX



This nearly exactly the same as the Main form in the design section. Since, it is already explained there, I don't see the need to document it again.

User Manual

CONTENTS

1. Introduction 3

2. Installation 3

3. The Different windows 4

Function transformations 4

Options box 5

Page setup dialog 6

Printer dialog 7

4. Transforming the graph of a function 8

Selecting a function 8

Function transformations with magnitude 9

Function transformations without magnitude 10

5. Changing the points that are highlighted 13

6. Displaying the Options box 13

7. Printing the graph 15

8. Reseting the graph 16

1. INTRODUCTION

This system is designed to aid in the teaching and understanding of the transformations of the graphs of mathematical functions. It is called "Graph Transformations" and it is mainly for use by Mathematics Teachers.

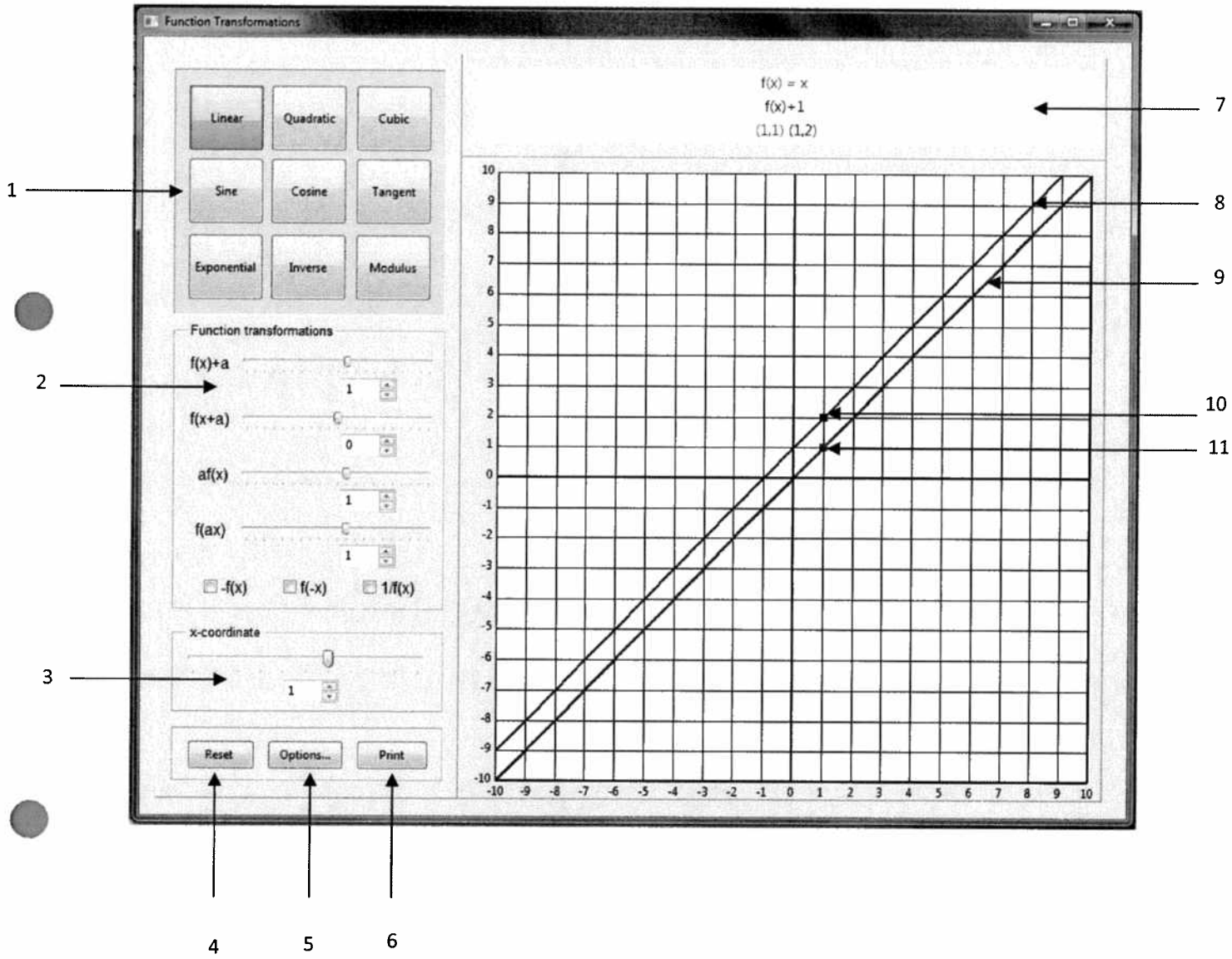
The main feature of the program is the ability to plot and display the graph of a selected mathematical function that has been transformed through the use of the application's interface. The program can also simultaneously display the graph of the original function on the same axes as the transformed graph, allowing direct comparison. In addition, two points with the same x-coordinate will be highlighted on the original and transformed graphs and their Cartesian coordinates will be displayed. This allows the user to see the effect of the transformation(s) on a specific point of the function. A mathematical description of the transformations incurred on the original graph is also displayed and the user has the option to reset the transformed graph to its original state. The User can choose to customise the graph and if required, print the graph.

2. INSTALLATION

The application is supplied via USB stick. No installation is required for this program as it can be directly executed by double-clicking on the icon, since it is an application.

3. THE DIFFERENT WINDOWS

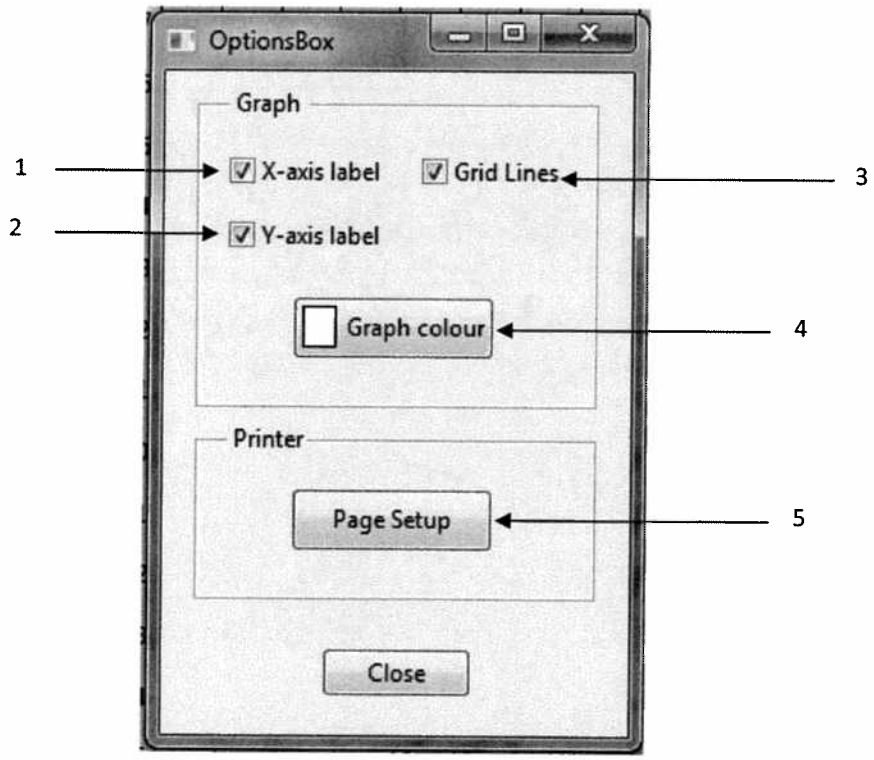
FUNCTION TRANSFORMATIONS



1. **Function selection panel** – This is where you chose your mathematical function
2. **Function transformation panel** – Contains sliders, textboxes and tick boxes to allow for transformation of transformed plot
3. **X-coordinate panel** – You can vary this to change the x coordinate of the highlighted point
4. **Reset button** – Use this after you have performed function transformations and wish to quickly return to original settings
5. **Options button** – This opens up the options box when pressed

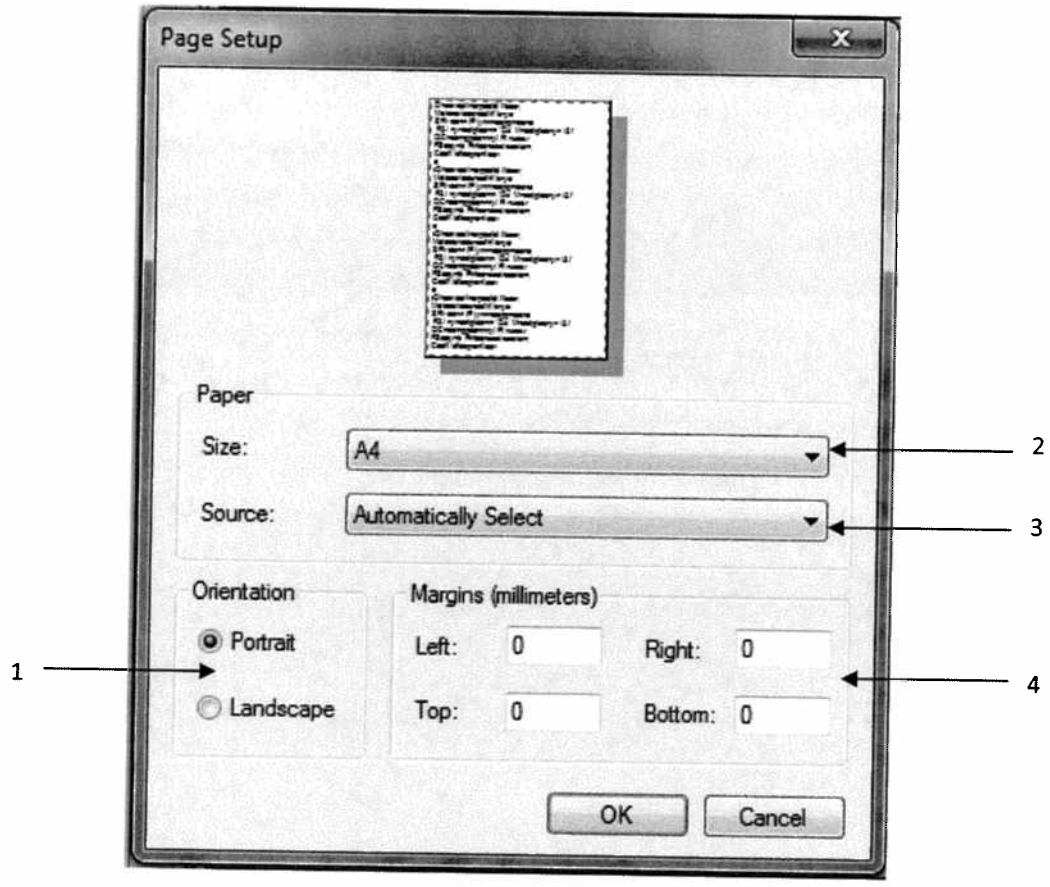
6. **Print button –** Click this when a print out of the currently drawn graph is required
7. **Text Information –** This displays the function currently selected and the effects of the currently applied transformations in Mathematical form. The Cartesian coordinates of the highlighted points are also displayed. The information is colour coded, with red relating to the original graph and blue corresponding to the transformed one.
8. **Plot of transformed function –** This is always going to be in blue and is the one that changes according to the function transformations chosen.
9. **Plot of Original function –** This red plot represents the untransformed function and will always remain in the same position, no matter what function transformations are applied.
10. **Highlighted point on transformed graph –** This is the point with the x-coordinate specified, it moves according to the transformation incurred onto the graph.
11. **Highlighted point on original graph –** This is the point on the original graph that will move according to the x-coordinate selected by the user.

OPTIONS BOX



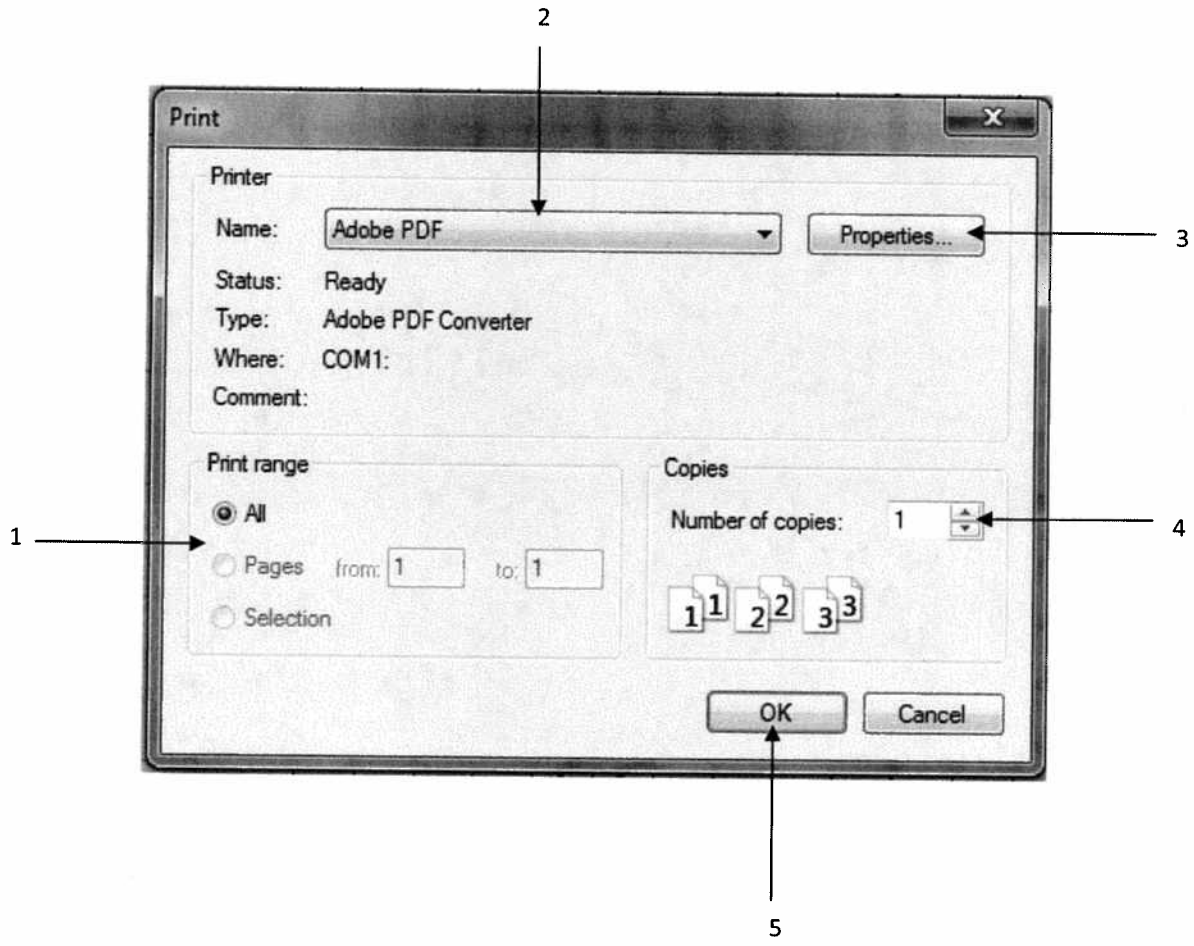
1. **X-axis label option** – Chose to turn on numbering of X-axis scale on main graph transformation window
2. **Y-axis label option** – Chose to turn on numbering of Y-axis scale on main graph transformation window
3. **Grid lines** – Chose to turn on grid lines on function transformations window
4. **Graph colour** – Opens up colour selection tool so you can chose your own colour for the background of the graph area in the Function Transformation window
5. **Page Setup Button** – Opens up Page Setup dialog so you can change settings of the page before printing the graph.

PAGE SETUP DIALOG



1. **Orientation of print page** – Select whether you want the graph to be printed landscape or portrait
2. **Size of print page** – Chose the type of page the graph will be printed on
3. **Page source** – For those printers that have more than one tray, this setting defines the tray to select pages from
4. **Page margins** – This sets the left, right, top and bottom page margins

PRINTER DIALOG

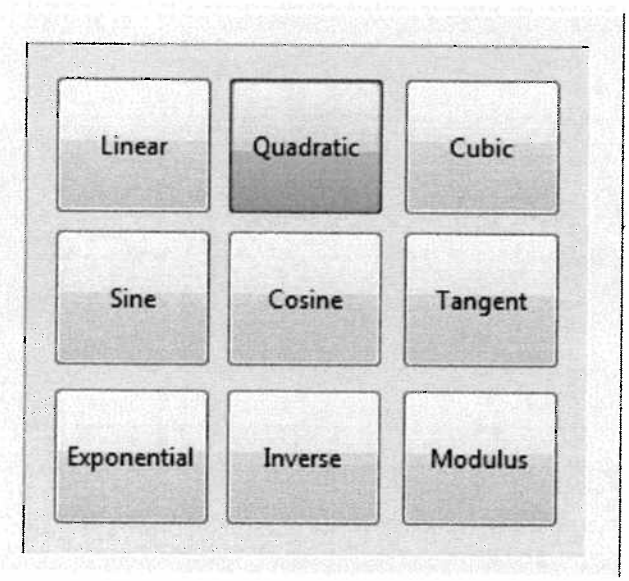


1. **Print range –** This will always be set to 'all' as there is only one page that will be printed, since the graph will always encompass one page
2. **Printer selection –** You can chose the printer that is going to print from here
3. **Printer properties –** The clicking of this button opens up a window for the properties of the printer currently selected
4. **Number of copies –** This specifies the number of copies to be printed of the currently displayed graph
5. **Ok button –** By clicking this the print job is executed and the printer starts printing the graph

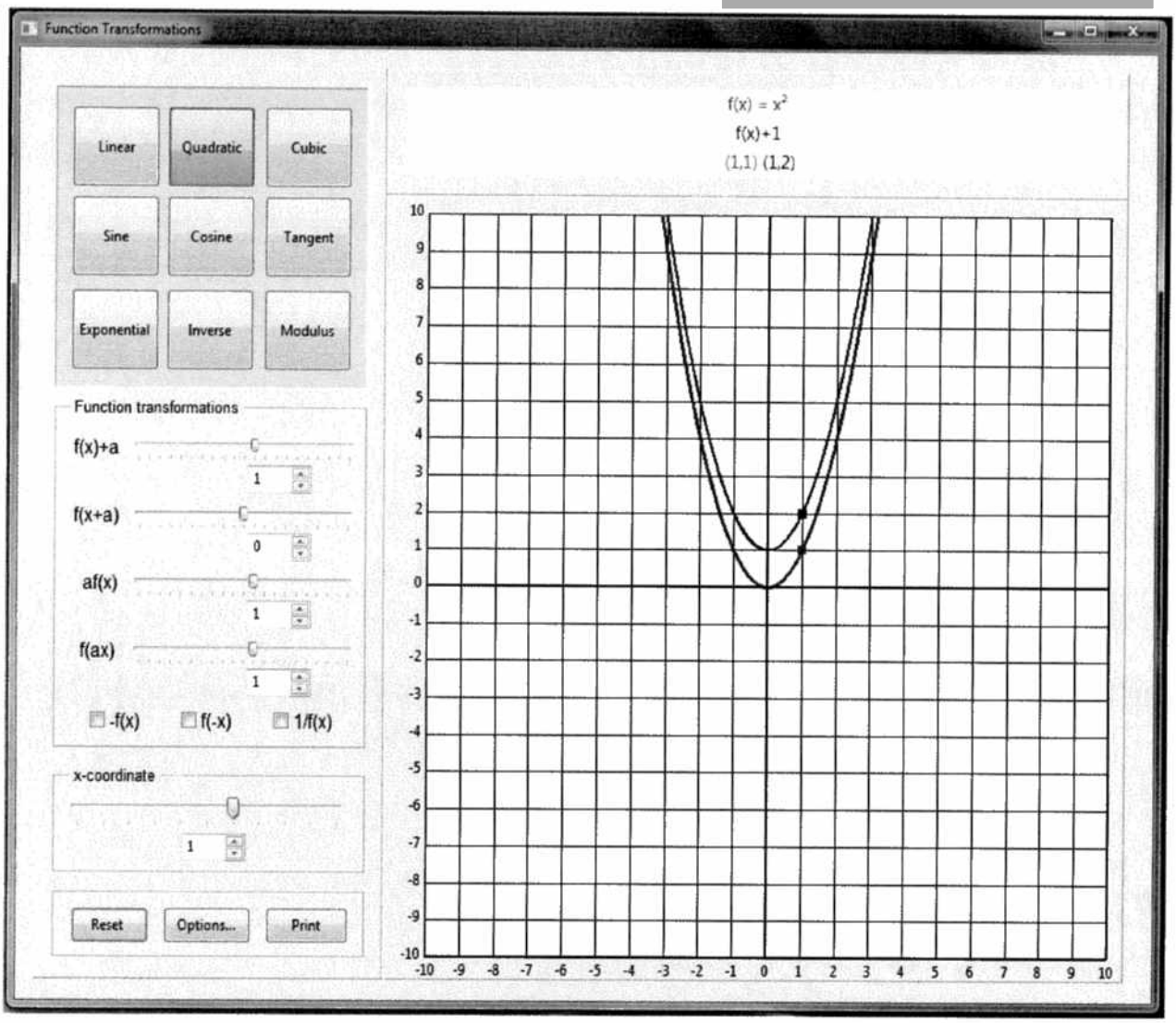
4. TRANSFORMING THE GRAPH OF A FUNCTION

SELECTING A FUNCTION

In order to carry out a function transformation it is necessary to first select the mathematical function to be plot. This can be done selecting the appropriate button from the "Function selection panel" which has its buttons arranged in the form of a keypad, as shown below:



After the function has been selected two graphs are shown - the original, untransformed graph of the selected function in red and a plot of the transformed graph in blue (which has already been transformed with initially chosen transformations). See below:



After this you can chose to incur one of the following transformations on the blue transformed graph:

FUNCTION TRANSFORMATIONS WITH MAGNITUDE

These types of transformations include the $[f(x) + a]$, $[f(x + a)]$, $[af(x)]$ and $[f(ax)]$ types and have magnitudes which can be altered. In my program the magnitudes can be altered in one of two ways: by entering a whole number value into a textbox with arrows (Figure1) or by varying the tab of a slider (Figure2).

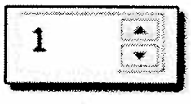


Figure 1

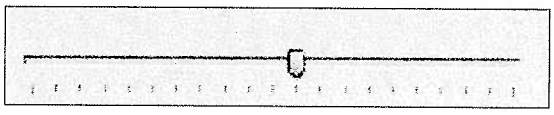
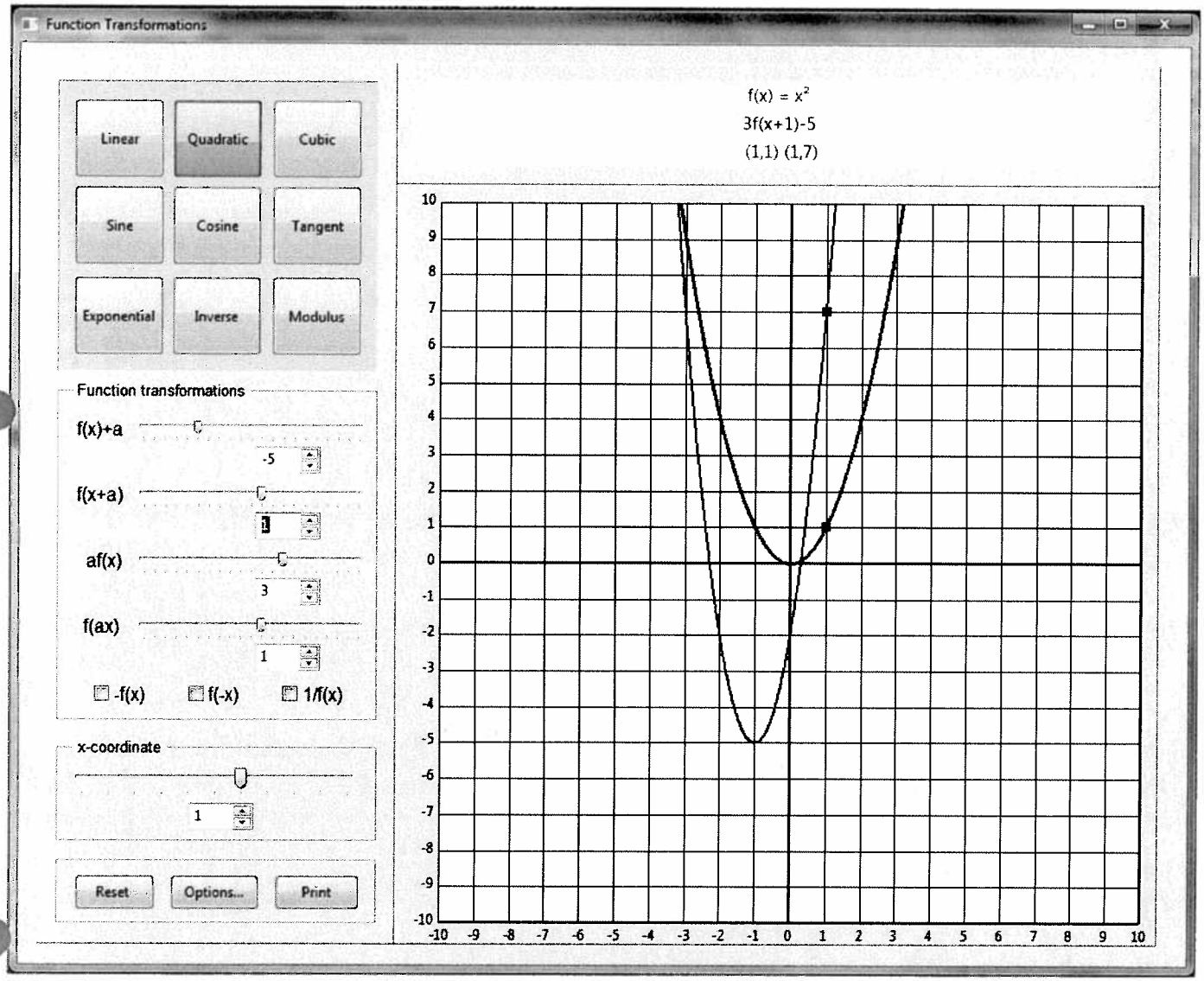


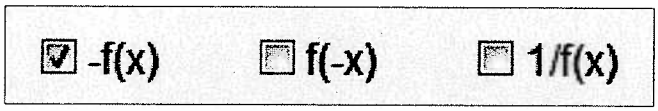
Figure 2

You aren't just restricted to applying only one transformation at once but many can be applied at the same time, as demonstrated below:



FUNCTION TRANSFORMATIONS WITHOUT MAGNITUDE

These types of transformations include the $[-f(x)]$, $[f(-x)]$ and $[\frac{1}{f(x)}]$ types and can only be turned on or off. Thus, to select their state tick boxes are used (see below).



You aren't restricted to applying only one of these sorts of transformation at once. More than one (including those with magnitudes) can be applied at the same time (see figure3).

However there is one exception, the $[\frac{1}{f(x)}]$ transformation can only be applied by itself, not with others (see figure4).

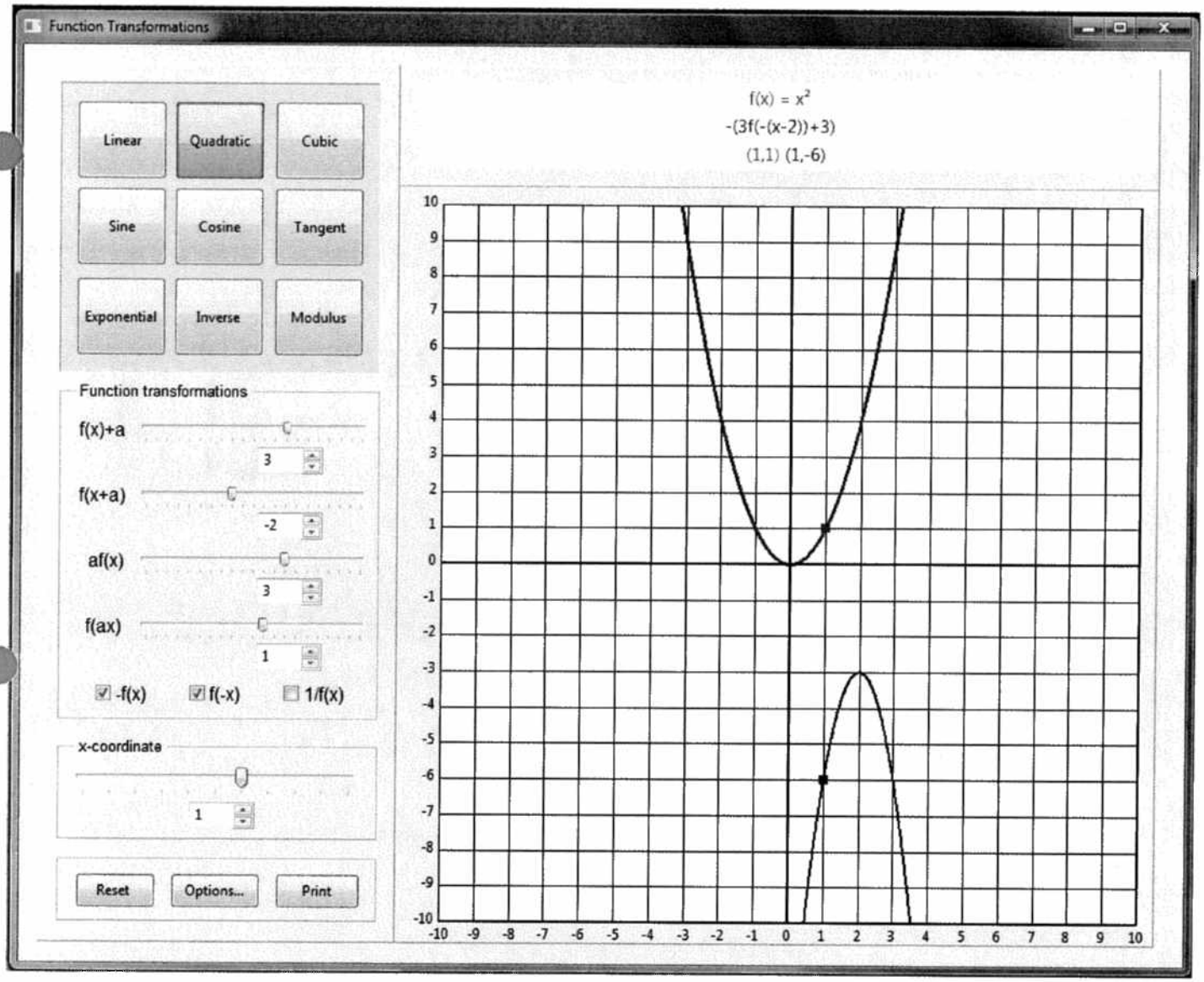


Figure 3

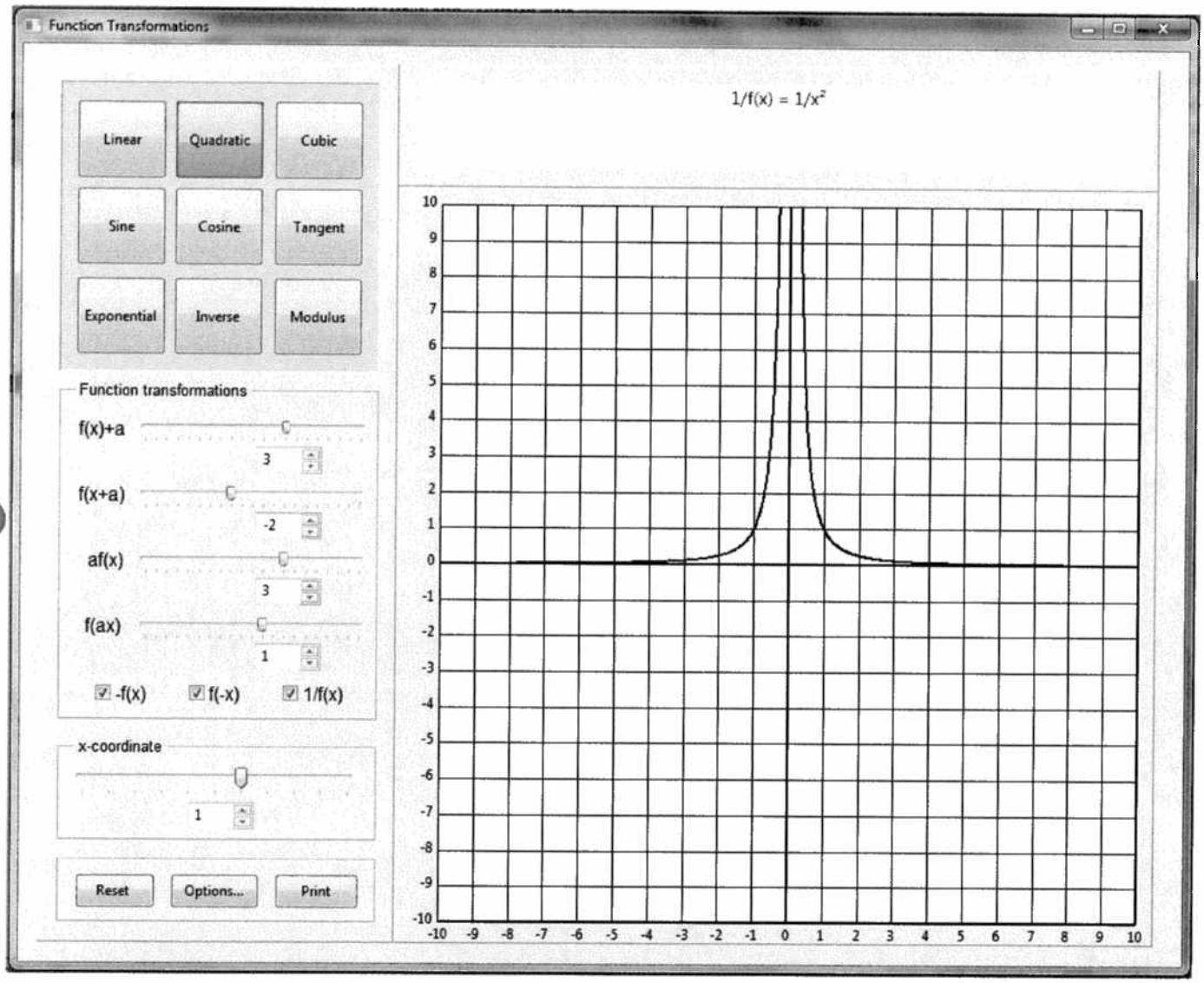
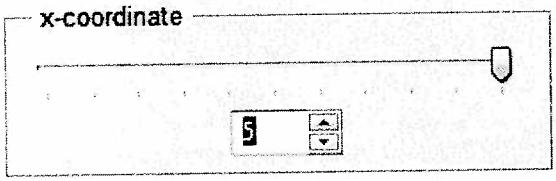


Figure 4

As you can see, when the $[\frac{1}{f(x)}]$ transformation is applied the plot changes colour to maroon.

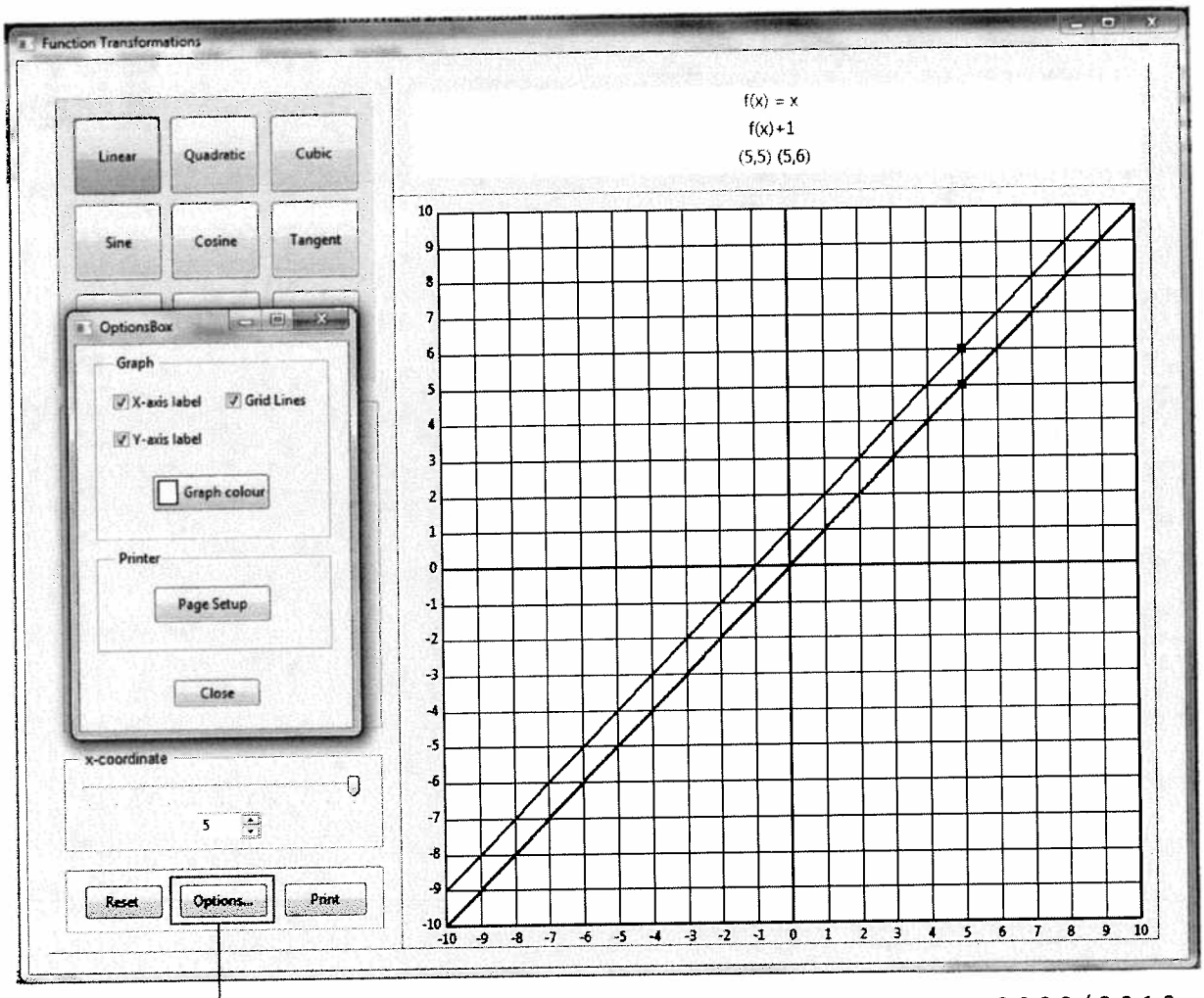
5. CHANGING THE POINTS THAT ARE HIGHLIGHTED

The X-coordinate of the points highlighted on both the original and transformed graphs can be changed by varying the slider or entering a value into the textbox of the X-coordinate panel shown below:



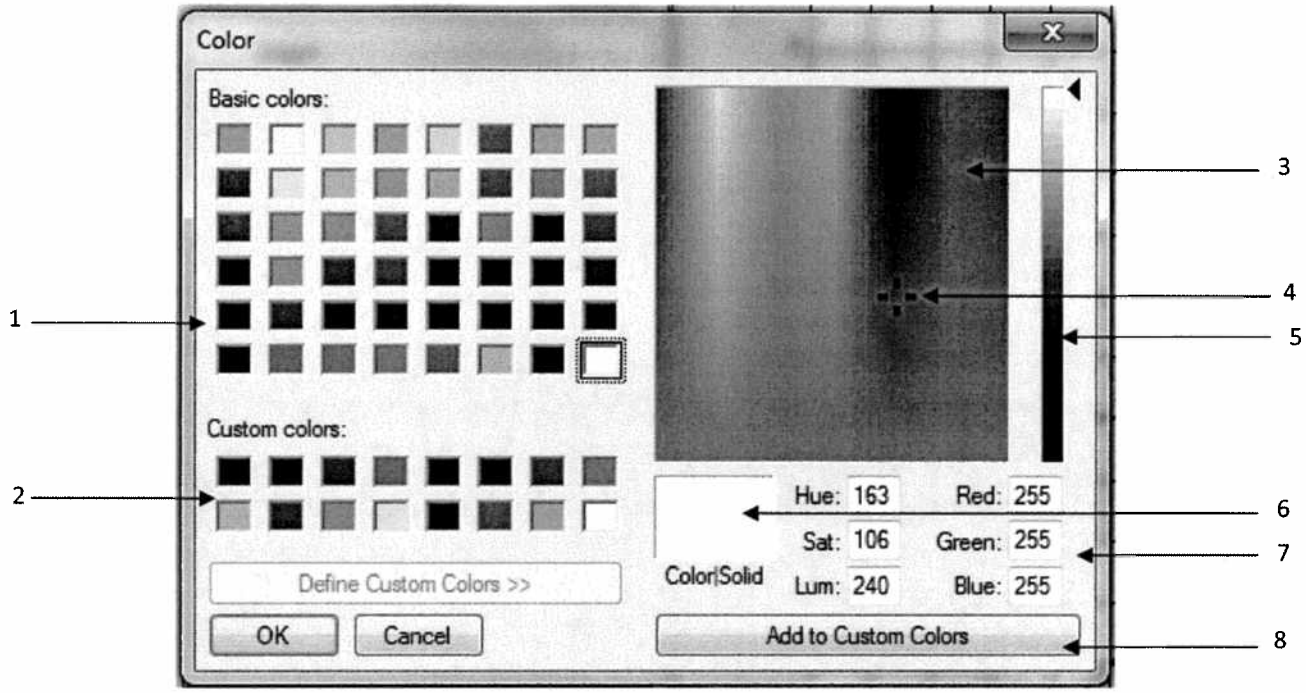
6. DISPLAYING THE OPTIONS BOX

The options box allows further customisation of the graph and selection of the print page settings. In order to display the options box the options button on the 'Function Transformations' window must be pressed (see below)



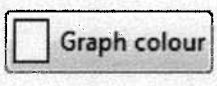
Press this to access Options

The tick boxes are used to make the X Labels, Y Labels and Grid lines visible on the graph. The Graph colour button when pressed, displays a colour selector dialog like the one below:

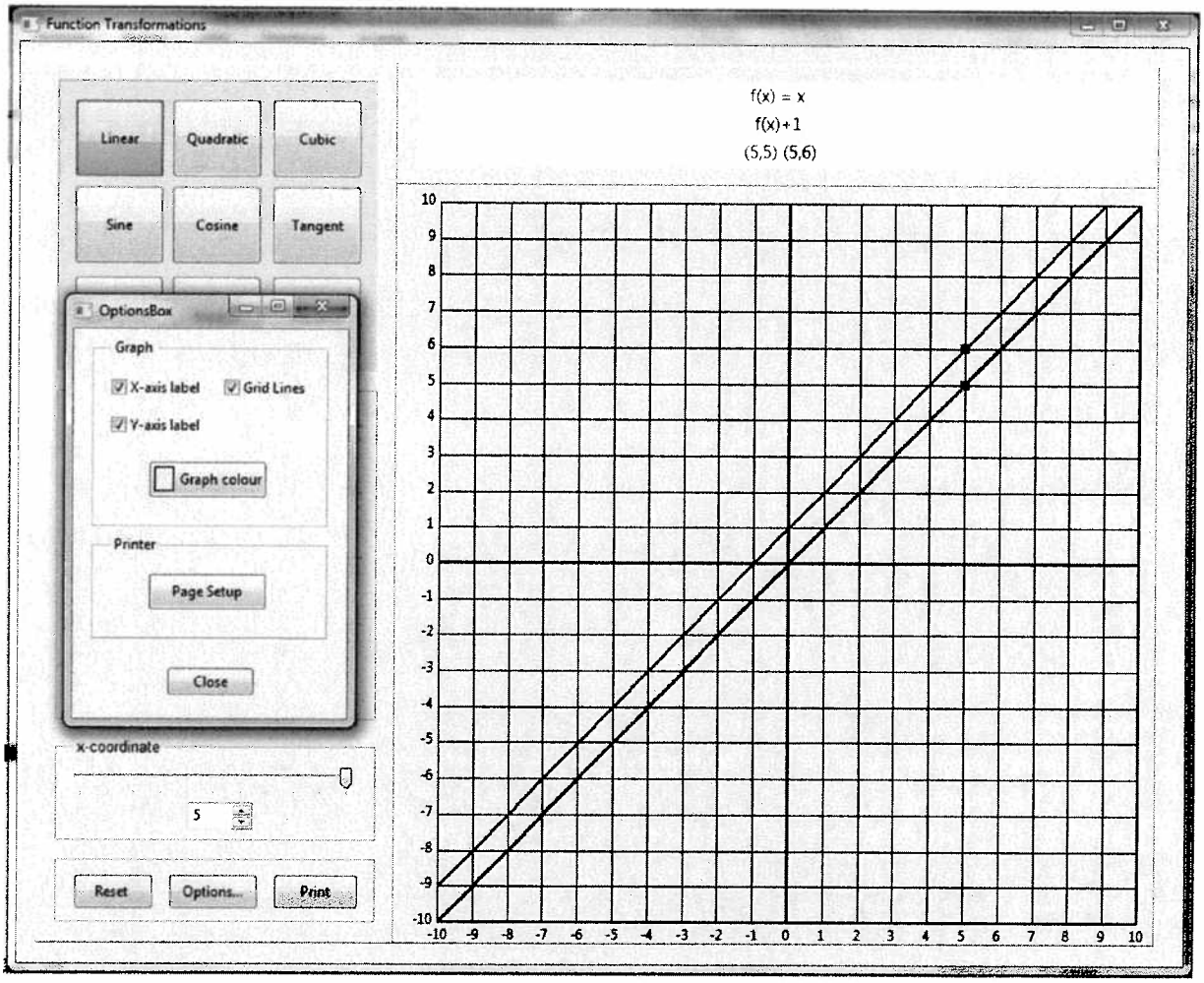


1. **Basic Colours :** These are preset colours which can't be changed
2. **Custom colours:** These are colours which can be created by defining properties on the bottom right of the dialog
3. **Colour creator window**: Any colour within the window can be selected using the pointer, and then that colour's specific properties will be shown in the textboxes below
4. **Pointer:** This is used to select a colour within the 'colour creator window'
5. **Contrast bar:** This is used to vary the contrast of the colour that is currently selected by the 'pointer'
6. **Colour|Solid box:** This displays the resulting colour after all the properties have been applied to it.
7. **Colour properties:** These are properties which can be adjusted in order to define a new colour
8. **Add to custom colours button:** This adds the colour currently shown in the 'Colour|Solid' window to the 'custom colours'

Whatever colour is chosen in the end, it is set as the background colour of the graph. For instance if the following colour were chosen:

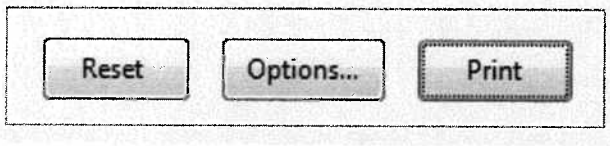


The result would be as follows:



7. PRINTING THE GRAPH

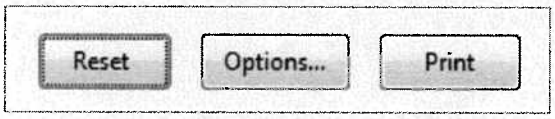
The whole process of printing the graph is initiated by pressing the print button on the bottom-left hand corner of the 'Function Transformations' window (as shown below).



When the button is clicked, the printer dialog shown in **section 3** ('The different windows') will appear. Using this dialog you will have to specify the printer and the amount of copies required. Once this is done, press OK to start printing.

8. RESETING THE GRAPH

In order to reset the transformed graph back to its initial transformation values, you must press the reset button on the bottom-left hand corner of the 'Function Transformations' window (as shown below).



Appraisal

Specification excerpt

- Evaluate methods and solutions against the specification and on the basis of effectiveness, usability and maintainability

CONTENTS

1. Comparison against original objectives 3

Input, processing and output requirements 3

performance requirements 5

2. User Feedback 5

3. Analysis of user feedback 6

4. Improvements required 6

5. Future extensions 6

1. COMPARISON AGAINST ORIGINAL OBJECTIVES

INPUT, PROCESSING AND OUTPUT REQUIREMENTS

1. The system must be able to plot and output the original and transformed graphs on the same axes of 9 different mathematical functions. Following on, the respective x and y axes must also have a range of at least -10 to 10 each.

Result

This objective is fulfilled as my system is able to plot the original and transformed graphs on the same axes of 9 different functions. The functions being the Linear, Quadratic, Cubic, Sine, Cosine, Tangent, Exponential, Inverse and Modulus ones. The ranges for the respective x and y axes also have a range from -10 to 10, except in the case of the trigonometric functions (Sine, Cosine and Tangent). This is because their x-values are in degrees and the x axis required to plot them, ranges from of -360 to 360 degrees.

2. The system must be able to produce seven different types of function transformations: $[f(x) \pm a]$, $[f(x \pm a)]$, $[af(x)]$,$[f(ax)]$, $[-f(x)]$, $[f(-x)]$ and $[\frac{1}{f(x)}]$.

Result

This objective is fulfilled as my system is able apply any of the seven function transformations.

3. For transformations that can have varying magnitudes such as $[f(x \pm a)]$, the range of the variable $'a'$ must range from -10 to 10.

Result

This objective is fulfilled as all the transformations with magnitudes, $[f(x) \pm a]$, $[f(x \pm a)]$, $[af(x)]$ and $[f(ax)]$, have a slider and textbox range of -10 to 10

4. The system must be able to produce text describing the effect and magnitude (if applicable) of the type of transformation/s currently applied on the selected function. The text must also change the value of the magnitude it states as the user varies it accordingly.

Result

This objective is fulfilled as my system successfully produces mathematical text to describe what effect the applied transformations have had on the selected function. Also, the text updates accordingly as the function transformations are varied.

5. The system must be able to highlight a point on the plots of both the original and transformed graphs corresponding to the user's selection and also output the Cartesian coordinates of that point.

Result

This objective is fulfilled as the system can highlight a point on the plots of both the original and transformed graphs for nine different functions. It can also output the Cartesian Coordinates of the highlighted points for all nine functions as well.

6. The system must be able allow the user to print a screenshot of the graph area

Result

This object remains unfulfilled as the user can print a screenshot of the graph area, but it is too small to see. This requirement has not been fulfilled due to lack of knowledge of coding and ultimately due to time constraints on the project.

PERFORMANCE REQUIREMENTS

1. The plot of the transformed graph must update smoothly and without flicker as the magnitude of the transformation is varied.

Result

This objective is fulfilled as the transformed graph is updated smoothly, in a flicker-free manner after a function transformation is applied.

2. For those transformations without magnitude the plot should update in less than two seconds when the transformation is applied (e.g. after a tick-box corresponding to the transformation is clicked).

Result

This objective is fulfilled as all the non-magnitude transformations update far less than 2 seconds after the transformation is applied/deactivated.

2. USER FEEDBACK

See Appendix Section 1.4 on User Evaluation

3. ANALYSIS OF USER FEEDBACK

TBA

4. IMPROVEMENTS REQUIRED

1. The Print function has to fixed, so that the whole graph is printed
2. Further testing could be carried out to the system, to identify the existence of any bugs
3. The code should be annotated
4. Efficiency could be increased by analyzing coding and deleting any unneeded repetitive coding

5. FUTURE EXTENSIONS

1. A quiz on transformations of graphs of functions, where a selection of questions are generated or chosen from a database. These questions could include matching up functions with graphs or identifying the type of transformation of graphs shown on screen.
2. Zoom in/out function. This allows the user to zoom into certain areas of the graph, meaning a variable x and y axis scale would be required. So as the user zooms in/out the scales would have to change their ranges.
3. In addition to having a set of nine functions to select, the user could also input any mathematical function of their own choice. The program could then plot this function and allow the user to transform it.
4. Make the project accessible for mathematical students as well as just teachers. This would mean that the description of what is happening to the graphs in terms of transformations have to be more thorough and easier to understand.

Bibliography and Appendix

APPENDIX CONTENTS

Bibliography 3

appendix 4

1.1 End user Questionnaire 4

1.2 Exsisting documentation 9

1.3 CCEA A-level Mathematics Specification excerpts 12

1.4 Coding Listing 14

Unit 1 14

OptionsBox 43

1.4 User evaluation 45

BIBLIOGRAPHY

Books

Smedley, R., & Wiseman, G. (2001). *Introducing Pure Mathematics.* Oxford: Oxford University Press.

Websites:

http://www.wikipedia.org/
http://en.wikipedia.org/wiki/Polymorphism_in_object-oriented_programming#Delphi
(Polymorphism coding techniques)
http://wiki.lazarus.freepascal.org/PlotPanel
(Help with PlotPanel)
http://www.delphibasics.co.uk/RTL.asp?Name=Class
(Creating objects within TForm)
http://www.delphibasics.co.uk/Article.asp?Name=Printing
(Printing text and graphics)
http://www.efg2.com/Lab/Library/Delphi/MathFunctions/General.htm
(Maths Functions)
http://www.aspfree.com/c/a/.NET/The-Delphi-Language-Part-2/3/
(circular Unit references)
http://etutorials.org/Programming/mastering+delphi+7/Part+I+Foundations/Chapter+7+Working+with+Forms/The+TForm+Class/
(Working with Forms)
http://www.delphibasics.co.uk/Article.asp?Name=Abstract
(Abstraction in object-orientated programming)
http://www.delphibasics.co.uk/RTL.asp?Name=Printer
(Information about the function, 'Printer')
http://www.delphibasics.co.uk/Article.asp?Name=Printing
(Printing text and graphics)
http://www.efg2.com/Lab/Library/UseNet/1999/0512f.txt
(print a bitmap snapshot of a virtual form)
http://www.delphitricks.com/source-code/math/round_numbers_to_a_predetermined_number_of_decimals.html
Get function to round decimals to predefined decimal places

APPENDIX

1.1 END USER QUESTIONNAIRE

A2 COMPUTING COURSEWORK QUESTIONNAIRE

For my Computing coursework I have decided to create a program which plots a variety of Mathematical functions, which varies the magnitude of certain types of transformation through user input. This is in a way is a tool to aid learning of the topic and make the transformations of graphs of functions occur in real-time. Here, I am going to ask you various questions about the current methods of teaching the topic and what you would like to see in the system I am designing:

SECTION 1: THE CURRENT SYSTEM:

1. Do Mathematics students find the topic of graphs of function transformations difficult to understand? (E.g. Would they in general lose marks in the topic during exams?)

Students find the transformation of functions one of the most difficult topics to understand. They find it difficult to work through the different types of transformations and frequently make mistakes.

2. To what level of Mathematics students would the topic of transformations of functions be taught? (e.g. to Key Stage 3, A-level, GCSE, etc.)

The transformation of functions is first introduced at GCSE level, although it is not referred to as the transformation of functions until the topic is studied in depth at A2 level.

3. Can you give a brief account of the current methods employed by teachers in teaching the topic? (Please state all and indicate which one is most commonly used)

Teachers explain the theory behind the transformation of functions using written notes. These notes are then supplemented using examples either drawn on the board, or more commonly displayed using Autograph software.

4. Do you feel there are any problems with the current methods? If yes, please state.

The examples need to be more active and dynamic, showing the different types of transformations of functions.

5. If you answered no to the previous question, are there aspects of the process you find inefficient or time consuming which could be improved?

—

6. If I developed a fully-functioning program that displayed real-time transformations of functions, what exactly will the system be used to achieve?

A system like this would be beneficial to the students and their understanding of the topic.

7. If I developed a fully-functioning program, would usage only be limited to teachers or would students be allowed to use as part of an interactive Math IT session or something similar?

Usage of such a program by the maths department would be limited to teachers. Due to the CCEA A-Level Maths specification, we do not have an interactive Maths IT session for A-Level Maths students.

SECTION 2: POSSIBLE SOLUTIONS

1. Apart from teaching the topic using the whiteboard, textbook or pre-written notes, what other ways are used to show the transformations that occur? For instance, would a school software package such as *Omnigraph* be used?

Autograph software.

2. If you gave different methods in the previous question, could you state what you find are the problems or disadvantages associated with each one?

See previous answer to question 4

SECTION 3: INPUTS

1. What would be the main functions whose graphs should be plotted and transformed?

Linear, quadratic and cubic functions.

2. What sort of function transformations would you require?

$f(x)+a$, $f(x+a)$, $af(x)$, $f(ax)$, $\frac{1}{f(x)}$

SECTION 4: OUTPUTS

1. Would you like to me include the initial untransformed graph in the background of the transformed plot as a default feature, or make it optional?

Yes

2. Should text be included to explain the nature of transformation/s? If yes please give an example of the detail required.

Yes, an explanation would be needed of how the general shape of a graph is transformed and how the x and y coordinates change.

3. Apart from the things I have suggested, is there anything else you would like to be displayed?

Labels on both the original and transformed graph.

4. Would you need print-outs of any plots from the program?

Yes, that would be useful

SECTION 5: PROCESSES AND FUNCTIONS

1. Are there any processes you would like me to perform on the functions or their plots that I haven't mentioned?

No

SECTION 6: ANY FURTHER IDEAS

1. Do you have any ideas or suggestions you would like to mention?

☺ THANK YOU FOR YOUR TIME ☺

1.2 EXSISTING DOCUMENTATION

TRANSFORMATION OF FUNCTIONS

$f(x)$ means a "function of x ", eg. $f(x)=2x$, $f(x)=x^2+4$.
There are different ways in which a function can be transformed:
(i) $f(x)\pm k$, (ii) $f(x\pm k)$, (iii) $-f(x)$, (iv) $kf(x)$, (v) $f(kx)$, (vi) $\frac{1}{f(x)}$.
k is just a constant.

(i)
$y=f(x)+k$ - the curve $f(x)$ is moved up by k units
$y=f(x)-k$ - the curve $f(x)$ is moved down by k units
eg.

(ii)
$y=f(x+k)$ - the curve $f(x)$ is moved k units to the left
$y=f(x-k)$ - the curve $f(x)$ is moved k units to the right
eg.

(iii)
$y=-f(x)$ - the curve $f(x)$ is reflected in the x-axis
eg.

(iv)
$y=kf(x)$ - the curve $f(x)$ is stretched parallel to the y-axis with scale factor k
eg.

(v)
$y=f(kx)$ - the curve $f(x)$ is stretched parallel to the x-axis with scale factor $\frac{1}{k}$
eg.

(vi)
$y=\frac{1}{f(x)}$ - when • $f(x)$ is large, $\frac{1}{f(x)}$ is small
• $f(x)$ has a maximum value, $\frac{1}{f(x)}$ has a minimum value
• $f(x)$ has a minimum value, $\frac{1}{f(x)}$ has a maximum value
$\frac{1}{f(x)}$ has the same sign as $f(x)$.

Compound Transformations

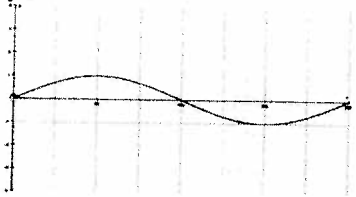
When combining transformations you should do the stretch/compression/reflection before a translation.

Horizontal transformations: To get from y = f(x) to y = f(ax + b) we must write it as $f(a(x+\frac{b}{a}))$ to determine the 'size' of the translation.

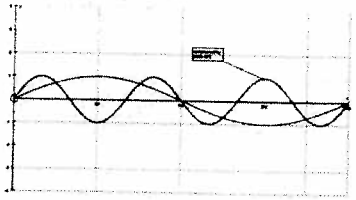
Ex 1/. Sketch y = sin (3x – 90) from y = sin x.

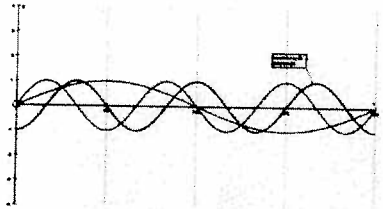
Note this can be written as y = sin [3(x-30)] so starting with y = sin x we compress by a factor of 3 and then translate to the right by 30 units:

y=sin x



compress by factor of 3

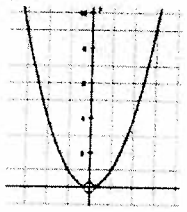




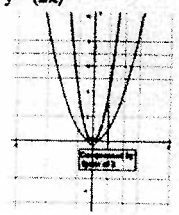
Ex 2/. Sketch $y = (2x-6)^2$ from $y = x^2$.

This we can write as $y = [(2(x-3)]^2$ so from $y = x^2$ we compress by a factor of 2 and then translate to the right by 3 units.

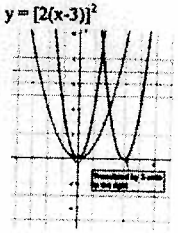
$y = x^2$



$y = (2x)^2$



$y = [2(x-3)]^2$



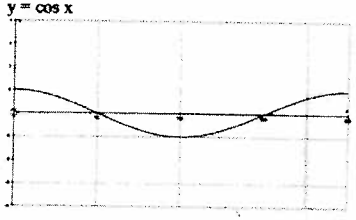
It is possible to do the translation before the stretch/compression/reflection however this requires you to realise that the invariant line has been translated before you can do the 2nd transformation. Consequently it is easier to make a mistake!

Vertical transformations: To get from y = f(x) to y = af(x) + b.

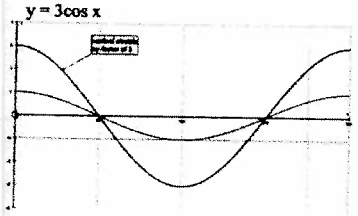
Ex 3/. Sketch y = 3cos x + 2 from y = cos x.

This is more straightforward. We firstly compress by a factor of 3 and then translate up 2 units.

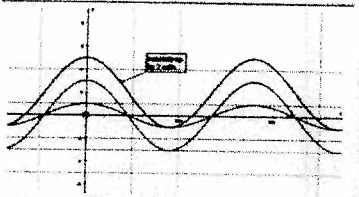
y = cos x



y = 3cos x



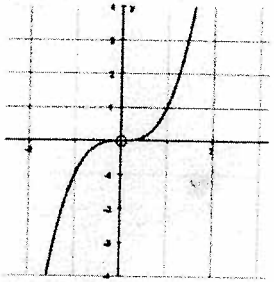
y = 3 cos x + 2



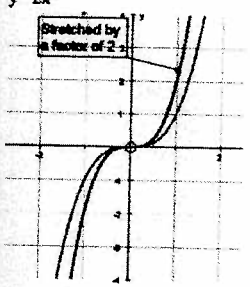
Ex 4/. Sketch $y = 2x^3 - 1$

Stretch vertically by a factor of 2 and then translate down by 1 unit.

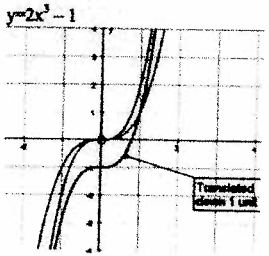
$y=x^3$



$y= 2x^3$



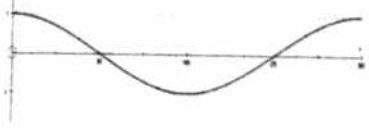
$y=2x^3 - 1$



Note:
If combining horizontal & vertical transformations it doesn't matter which you do first as long as you stick with the seperate rules for Horizontal & Vertical transformations already given!

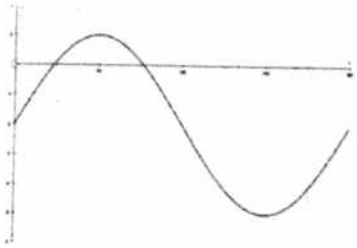
COMPOUND TRANSFORMATIONS

1/. $y = \sin(90 - x)$

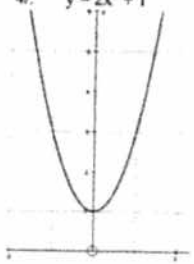


2/. $y = \tan(90 - 2x)$

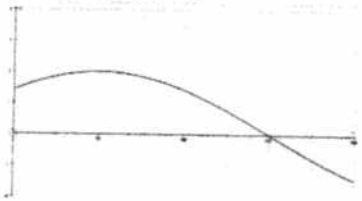
3/. $y = 3\sin x - 2$



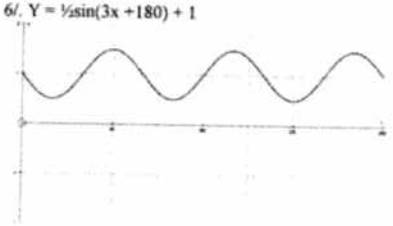
4/. $y = 2x^2 + 1$



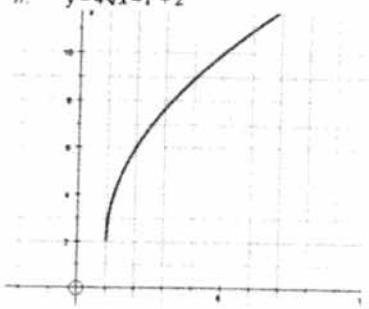
5/. $y = 2\cos(\frac{1}{2}x - 45)$



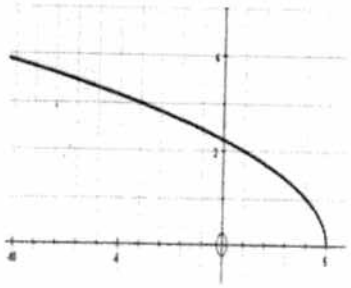
6/. $Y = \frac{1}{2}\sin(3x + 180) + 1$



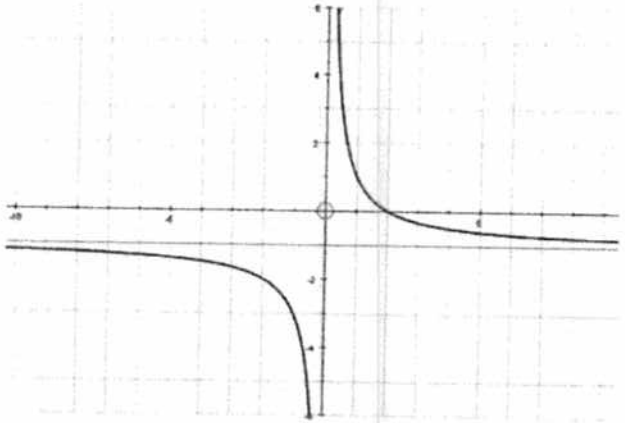
7/. $y = 4\sqrt{x-1} + 2$



8/. $y = \sqrt{5 - x}$



9/. $y = \frac{2}{x} - 1$



1.3 CCEA A-LEVEL MATHEMATICS SPECIFICATION EXCERPTS

MODULE C1 – AS CORE MATHEMATICS 1

This module covers approximately half of the core content material for AS examinations. The module will be assessed at AS standard and is compulsory for AS and A level GCE Mathematics. The assessment unit for this module is an external examination, with a maximum of 75 raw marks, the duration of which is 1 hour 30 minutes. **Candidates are not permitted to use any calculating aid in the assessment unit for this module.**

	Topic	Guidance Notes
1	Laws of indices for all rational indices.	
	Use and manipulation of surds.	Rationalisation of denominators
2	Quadratic functions and their graphs; the discriminant of a quadratic function; completing the square.	Conditions for real and equal roots are included. Excluding relationship between roots and coefficients of a quadratic equation.
	Solution of quadratic equations.	
	Simultaneous equations; analytic solution by substitution, eg of one linear and one quadratic equation.	Linear with two or three unknowns.
	Solution of linear and quadratic inequalities.	Including inequalities reducible to the form $f(x) > 0$, where $f(x)$ is a product of linear factors.
3	Algebraic manipulation of polynomials, including expanding brackets and collecting like terms, factorisation and simple algebraic division.	Division by linear expressions only.
	Use of the Factor Theorem and the Remainder Theorem.	
4	Graphs of functions; sketching curves defined by simple equations.	Knowledge of function notation. Plotting graphs on graph paper will not be required.
	Geometrical interpretation of algebraic solution of equations. Use of intersection points of graphs to solve equations.	
	Knowledge of the effect of simple transformations on the graph of $y = f(x)$ as represented by $y = af(x)$, $y = f(x) + a$, $y = f(x + a)$, $y = f(ax)$.	
5	Equation of a straight line, including the forms $y - y_1 = m(x - x_1)$ and $ax + by + c = 0$.	Including the mid-point of a line segment.
	Conditions for two straight lines to be parallel or perpendicular to each other.	

MODULE C3 – A2 CORE MATHEMATICS 1

In following a course based on this specification students should be encouraged to make appropriate use of 'graphic' calculators and computers as tools by which the learning of mathematics may be enhanced. This module covers approximately half of the core content material for A level examinations beyond the AS core material contained in modules C1 and C2. A knowledge of the content of modules C1 and C2 will be assumed. The module will be assessed at A2 standard and is compulsory for A level GCE Mathematics. The assessment unit for the module is an external examination, with a maximum of 75 raw marks, the duration of which is 1 hour 30 minutes. The use of a 'graphic' or 'scientific' calculator will be permitted in the assessment unit for this module.

	Topic	**Guidance Notes**
1	Simplification of rational expressions including factorizing and cancelling, and algebraic division.	Division by non-linear expressions.
	Rational functions; partial fractions with denominators not more complicated than repeated linear terms.	
	The modulus function.	Including $\|x - a\| < b$
	Combinations of simple transformations on the graph of $y = f(x)$ as represented by $y = af(x)$, $y = f(x) + a$, $y = f(x + a)$, $y = f(ax)$.	
2	Parametric equations of curves; conversion between parametric and Cartesian forms.	General properties of conics are excluded.
3	Binomial series for any rational value of n.	
4	Knowledge of secant, cosecant and cotangent and of arcsin, arccos and arctan. Their relationships to sine, cosine and tangent.	Use of notation $\sin^{-1}\theta$ etc.
	Knowledge and use of the equivalents of $\sin^2\theta + \cos^2\theta = 1$.	$\sec^2\theta = \tan^2\theta + 1$ $\operatorname{cosec}^2\theta = 1 + \cot^2\theta$
		Solution of trigonometric equations in a given interval; for example, $2\sec^2\theta + 5\tan\theta = 5$.
5	The function e^x and its graph.	Knowledge of the effect of simple transformations.
	The function $\ln x$ and its graph; $\ln x$ as the inverse function of e^x.	Knowledge of the effect of simple transformations.
	Exponential growth and decay.	Both discrete and continuous growth. For example, the half-life of a radioactive element.

1.4 CODING LISTING

UNIT 1

```
unit Unit1;

{$mode objfpc}{$H+}

interface

uses
  Classes, StrUtils, SysUtils, FileUtil, LResources, Forms, Controls, Graphics,
  Dialogs, ComCtrls, ExtCtrls, StdCtrls, Menus, Math, ActnList, ButtonPanel, ExtDlgs,
  Spin, Plotpanel, PrintersDlgs, Printers, Unit2 {I coded Unit2}, Windows;

type

{ TForm1 }

 TForm1 = class(TForm)
   PrintButton: TButton;
   ResetButton: TButton;
   OptionsButton: TButton;
   ReflectX: TCheckBox;
   ReflectY: TCheckBox;
   InverseX: TCheckBox;
   FunctionsKeyPad: TPanel;
   ButtonPanel: TPanel;
   ToolPanel: TPanel;
   TextPanel: TPanel;
   FunctTransPanel: TPanel;
   XCoordPanel: TPanel;
   YTransEdit: TSpinEdit;
   XTransEdit: TSpinEdit;
   YStretchEdit: TSpinEdit;
   XStretchEdit: TSpinEdit;
   XCoordEdit: TSpinEdit;
   YTransTxt: TStaticText;
   XCoordLabel: TStaticText;
   FunctTransLabel: TStaticText;
   TransFunctTxt: TStaticText;
   OrigFunctTxt: TStaticText;
   OrigCartCoord: TStaticText;
   TransCartCoord: TStaticText;
   XTransTxt: TStaticText;
   YStretchTxt: TStaticText;
   XStretchTxt: TStaticText;
   InverseFunction: TToggleBox;
```

```
    ModulusFunction: TToggleBox;
    PlotPanel1: TPlotPanel;
    LinearFunction: TToggleBox;
    SineFunction: TToggleBox;
    CosineFunction: TToggleBox;
    TangentFunction: TToggleBox;
    ExponentialFunction: TToggleBox;
    QuadraticFunction: TToggleBox;
    CubicFunction: TToggleBox;
    Ystretch: TTrackBar;
    XStretch: TTrackBar;
    XCoordinate: TTrackBar;
    YTranslation: TTrackBar;
    XTranslation: TTrackBar;
    procedure PrintButtonClick(Sender: TObject);
    procedure ReflectXClick(Sender: TObject);
    procedure ReflectYClick(Sender: TObject);
    procedure InverseXClick(Sender: TObject);
    procedure ResetButtonClick(Sender: TObject);
    procedure OptionsButtonClick(Sender: TObject);
    procedure CubicFunctionClick(Sender: TObject);
    procedure ExponentialFunctionClick(Sender: TObject);
    procedure InverseFunctionClick(Sender: TObject);
    procedure LinearFunctionClick(Sender: TObject);
    procedure ModulusFunctionClick(Sender: TObject);
    procedure QuadraticFunctionClick(Sender: TObject);
    procedure SineFunctionClick(Sender: TObject);
    procedure CosineFunctionClick(Sender: TObject);
    procedure YTransEditChange(Sender: TObject);
    procedure XTransEditChange(Sender: TObject);
    procedure YStretchEditChange(Sender: TObject);
    procedure XStretchEditChange(Sender: TObject);
    procedure XCoordEditChange(Sender: TObject);
    procedure TangentFunctionClick(Sender: TObject);
    procedure XCoordinateChange(Sender: TObject);
    procedure XStretchChange(Sender: TObject);
    procedure XTranslationChange(Sender: TObject);
    procedure YstretchChange(Sender: TObject);
    procedure YTranslationChange(Sender: TObject);
  private
    { private declarations }
  public
    { public declarations }
  end;

{ TFunction }

  TFunction = class
    Private
      Xmin : integer;
      Xmax : integer;
      Ymin : integer;
```

```
    Ymax : integer;
    a : integer;
    b : integer;
    c : integer;
    d : integer;
    e : integer;
    f  : integer; // -f(x)
    g : integer; // f(-x)
    InvX : boolean;  // 1/f(x)
    YValue : Extended;
    Sa, Sb, Sc, Sd  : string;
    TextInfo : String; //Static text in textbox
    XRed, XBlue, YRed, YBlue  : string;

  //Can't call PlotPanel1 here since it is already within the Form, so I must
  //continually reference it through parameters.
  Public
    Procedure SetXYAxesLimits(minXval,maxXval,minYval,maxYval: integer;
                  PlotPanel1: TPlotPanel);
    Procedure SetXYAxesIntervalsAndLayers (Xinter,Yinter: integer; PlotPanel1: TPlotPanel);
    Procedure PlotXYAxes(PlotPanel1: TPlotPanel);
    Procedure GetFunctionTransformationMagnitudes(Form1: TForm1);
    Procedure CalculateTransformedGraphInfo;
    Procedure DisplayCartCoordAndTransFunctTxt(Form1: TForm1);
    Procedure ToggleboxFix(Form1: TForm1); Virtual; Abstract;
    Procedure PlotFunction(PlotPanel1: TPlotPanel); Virtual; Abstract;
    Procedure HighlightPoints(PlotPane1: TPlotPanel);Virtual; Abstract;
    Procedure ApplyTransformation(PlotPanel1: TPlotPanel); Virtual; Abstract;
    Procedure DisplayOrigTxtAndCalcCartCoord(Form1: TForm1); Virtual; Abstract;

end;

TLinear = class(TFunction)
    Procedure ToggleboxFix(Form1: TForm1); Override;
    Procedure PlotFunction (PlotPanel1: TPlotPanel); Override;
    Procedure ApplyTransformation(PlotPanel1: TPlotPanel); Override;
    Procedure HighlightPoints(PlotPanel1: TPlotPanel); Override;
    Procedure DisplayOrigTxtAndCalcCartCoord(Form1: TForm1); Override;
    End;

TQuadratic = class(TFunction)
    Procedure ToggleboxFix(Form1: TForm1); Override;
    Procedure PlotFunction (PlotPanel1: TPlotPanel); Override;
    Procedure ApplyTransformation(PlotPanel1: TPlotPanel); Override;
    Procedure HighlightPoints(PlotPanel1: TPlotPanel); Override;
    Procedure DisplayOrigTxtAndCalcCartCoord(Form1: TForm1); Override;
    End;

TCubic = class(TFunction)
    Procedure ToggleboxFix(Form1: TForm1); Override;
    Procedure PlotFunction (PlotPanel1: TPlotPanel); Override;
    Procedure ApplyTransformation(PlotPanel1: TPlotPanel); Override;
```

```
    Procedure HighlightPoints(PlotPanel1: TPlotPanel); Override;
    Procedure DisplayOrigTxtAndCalcCartCoord(Form1: TForm1); Override;
    End;

  TSine = class(TFunction)
    Procedure ToggleboxFix(Form1: TForm1); Override;
    Procedure PlotFunction (PlotPanel1: TPlotPanel); Override;
    Procedure ApplyTransformation(PlotPanel1: TPlotPanel); Override;
    Procedure HighlightPoints(PlotPanel1: TPlotPanel); Override;
    Procedure DisplayOrigTxtAndCalcCartCoord(Form1: TForm1); Override;
    End;

  TCosine = class(TFunction)
    Procedure ToggleboxFix(Form1: TForm1); Override;
    Procedure PlotFunction (PlotPanel1: TPlotPanel); Override;
    Procedure ApplyTransformation(PlotPanel1: TPlotPanel); Override;
    Procedure HighlightPoints(PlotPanel1: TPlotPanel); Override;
    Procedure DisplayOrigTxtAndCalcCartCoord(Form1: TForm1); Override;
    End;

  TTangent = class(TFunction)
    Procedure ToggleboxFix(Form1: TForm1); Override;
    Procedure PlotFunction (PlotPanel1: TPlotPanel); Override;
    Procedure ApplyTransformation(PlotPanel1: TPlotPanel); Override;
    Procedure HighlightPoints(PlotPanel1: TPlotPanel); Override;
    Procedure DisplayOrigTxtAndCalcCartCoord(Form1: TForm1); Override;
    End;

  TExponential = class(TFunction)
    Procedure ToggleboxFix(Form1: TForm1); Override;
    Procedure PlotFunction (PlotPanel1: TPlotPanel); Override;
    Procedure ApplyTransformation(PlotPanel1: TPlotPanel); Override;
    Procedure HighlightPoints(PlotPanel1: TPlotPanel); Override;
    Procedure DisplayOrigTxtAndCalcCartCoord(Form1: TForm1); Override;
    End;

  TInverse = class(TFunction)
    Procedure ToggleboxFix(Form1: TForm1); Override;
    Procedure PlotFunction (PlotPanel1: TPlotPanel); Override;
    Procedure ApplyTransformation(PlotPanel1: TPlotPanel); Override;
    Procedure HighlightPoints(PlotPanel1: TPlotPanel); Override;
    Procedure DisplayOrigTxtAndCalcCartCoord(Form1: TForm1); Override;
    End;

  TModulus = class(TFunction)
    Procedure ToggleboxFix(Form1: TForm1); Override;
    Procedure PlotFunction (PlotPanel1: TPlotPanel); Override;
    Procedure ApplyTransformation(PlotPanel1: TPlotPanel); Override;
    Procedure HighlightPoints(PlotPanel1: TPlotPanel); Override;
    Procedure DisplayOrigTxtAndCalcCartCoord(Form1: TForm1); Override;
    End;
```

```
var
  Form1: TForm1;
  Function1: TFunction;

implementation

{ Non-Method Procedures }

Procedure UpdateTransformedGraph;
begin
  if Form1.LinearFunction.State = cbChecked
    then Function1:= TLinear.Create;
  if Form1.QuadraticFunction.State = cbChecked
    then Function1:= TQuadratic.Create;
  if Form1.CubicFunction.State = cbChecked
    then Function1:= TCubic.Create;
  if Form1.SineFunction.State = cbChecked
    then Function1:= TSine.Create;
  if Form1.CosineFunction.State = cbChecked
    then Function1:= TCosine.Create;
  if Form1.TangentFunction.State = cbChecked
    then Function1:= TTangent.Create;
  if Form1.ExponentialFunction.State = cbChecked
    then Function1:= TExponential.Create;
  if Form1.InverseFunction.State = cbChecked
    then Function1:= TInverse.Create;
  if Form1.ModulusFunction.State = cbChecked
    then Function1:= TModulus.Create;
  Function1.GetFunctionTransformationMagnitudes(Form1);
  Function1.ApplyTransformation(Form1.PlotPanel1);
  Function1.HighlightPoints(Form1.PlotPanel1);
  Function1.CalculateTransformedGraphInfo;
  Function1.DisplayOrigTxtAndCalcCartCoord(Form1);
  Function1.DisplayCartCoordAndTransFunctTxt(Form1);
  Function1.Free;
end;



{ TFunction }

Procedure TFunction.SetXYAxesLimits(minXval,maxXval,minYval,maxYval: integer;
                                    PlotPanel1: TPlotPanel);
Begin
  Xmin:= minXval;
  Xmax:= maxXval;
  Ymin:= minYval;
  Ymax:= maxYval;
  PlotPanel1.Freeze(True);
  PlotPanel1.XMin:= Xmin;
  PlotPanel1.XMax:= Xmax;
  PlotPanel1.YMin:= Ymin;
```

```
  PlotPanel1.YMax:= Ymax;
end;

Procedure TFunction.SetXYAxesIntervalsAndLayers(Xinter,Yinter: integer; PlotPanel1: TPlotPanel);
Begin
  PlotPanel1.XInterval:= Xinter;
  PlotPanel1.YInterval:= Yinter;
  PlotPanel1.LayerOptions(0,pmLine,3);
  PlotPanel1.LayerOptions(1,pmLine,2);
  PlotPanel1.LayerOptions(2,pmLine,2);
  PlotPanel1.LayerOptions(3,pmLine,2);
  PlotPanel1.LayerOptions(4,pmLine,2);
  PlotPanel1.LayerOptions(5,pmdot,8);
  PlotPanel1.LayerOptions(6,pmdot,8);
  PlotPanel1.ClearData;
End;


Procedure TFunction.PlotXYAxes (PlotPanel1: TPlotPanel);
Var
  i : integer;
Begin
  for i:= Xmin to Xmax
    do PlotPanel1.AddXY(i,0,clblack,1);
  for i:= Ymin to Ymax
    do PlotPanel1.AddXY(0,i,clblack,2);
end;

// ToggleboxFix

Procedure TLinear.ToggleboxFix(Form1: TForm1);
Begin
  Form1.QuadraticFunction.state:= cbUnchecked;
  Form1.CubicFunction.state:= cbUnchecked;
  Form1.SineFunction.state:= cbUnchecked;
  Form1.CosineFunction.state:= cbUnchecked;
  Form1.TangentFunction.state:= cbUnchecked;
  Form1.ExponentialFunction.state:= cbUnchecked;
  Form1.InverseFunction.state:= cbUnchecked;
  Form1.ModulusFunction.state:= cbUnchecked;
end;

Procedure TQuadratic.ToggleboxFix(Form1: TForm1);
Begin
  Form1.LinearFunction.state:= cbUnchecked;
  Form1.CubicFunction.state:= cbUnchecked;
  Form1.SineFunction.state:= cbUnchecked;
  Form1.CosineFunction.state:= cbUnchecked;
  Form1.TangentFunction.state:= cbUnchecked;
  Form1.ExponentialFunction.state:= cbUnchecked;
  Form1.InverseFunction.state:= cbUnchecked;
  Form1.ModulusFunction.state:= cbUnchecked;
```

```
end;

Procedure TCubic.ToggleboxFix(Form1: TForm1);
Begin
 Form1.LinearFunction.state:= cbUnchecked;
 Form1.QuadraticFunction.state:= cbUnchecked;
 Form1.SineFunction.state:= cbUnchecked;
 Form1.CosineFunction.state:= cbUnchecked;
 Form1.TangentFunction.state:= cbUnchecked;
 Form1.ExponentialFunction.state:= cbUnchecked;
 Form1.InverseFunction.state:= cbUnchecked;
 Form1.ModulusFunction.state:= cbUnchecked;
end;

Procedure TSine.ToggleboxFix(Form1: TForm1);
Begin
 Form1.LinearFunction.state:= cbUnchecked;
 Form1.QuadraticFunction.state:= cbUnchecked;
 Form1.CubicFunction.state:= cbUnchecked;
 Form1.CosineFunction.state:= cbUnchecked;
 Form1.TangentFunction.state:= cbUnchecked;
 Form1.ExponentialFunction.state:= cbUnchecked;
 Form1.InverseFunction.state:= cbUnchecked;
 Form1.ModulusFunction.state:= cbUnchecked;
end;

Procedure TCosine.ToggleboxFix(Form1: TForm1);
Begin
 Form1.LinearFunction.state:= cbUnchecked;
 Form1.QuadraticFunction.state:= cbUnchecked;
 Form1.CubicFunction.state:= cbUnchecked;
 Form1.SineFunction.state:= cbUnchecked;
 Form1.TangentFunction.state:= cbUnchecked;
 Form1.ExponentialFunction.state:= cbUnchecked;
 Form1.InverseFunction.state:= cbUnchecked;
 Form1.ModulusFunction.state:= cbUnchecked;
end;

Procedure TTangent.ToggleboxFix(Form1: TForm1);
Begin
 Form1.LinearFunction.state:= cbUnchecked;
 Form1.QuadraticFunction.state:= cbUnchecked;
 Form1.CubicFunction.state:= cbUnchecked;
 Form1.SineFunction.state:= cbUnchecked;
 Form1.CosineFunction.state:= cbUnchecked;
 Form1.ExponentialFunction.state:= cbUnchecked;
 Form1.InverseFunction.state:= cbUnchecked;
 Form1.ModulusFunction.state:= cbUnchecked;
end;

Procedure TExponential.ToggleboxFix(Form1: TForm1);
Begin
```

```
  Form1.LinearFunction.state:= cbUnchecked;
  Form1.QuadraticFunction.state:= cbUnchecked;
  Form1.CubicFunction.state:= cbUnchecked;
  Form1.SineFunction.state:= cbUnchecked;
  Form1.CosineFunction.state:= cbUnchecked;
  Form1.TangentFunction.state:= cbUnchecked;
  Form1.InverseFunction.state:= cbUnchecked;
  Form1.ModulusFunction.state:= cbUnchecked;
end;

Procedure TInverse.ToggleboxFix(Form1: TForm1);
Begin
  Form1.LinearFunction.state:= cbUnchecked;
  Form1.QuadraticFunction.state:= cbUnchecked;
  Form1.CubicFunction.state:= cbUnchecked;
  Form1.SineFunction.state:= cbUnchecked;
  Form1.CosineFunction.state:= cbUnchecked;
  Form1.TangentFunction.state:= cbUnchecked;
  Form1.ExponentialFunction.state:= cbUnchecked;
  Form1.ModulusFunction.state:= cbUnchecked;
end;

Procedure TModulus.ToggleboxFix(Form1: TForm1);
Begin
  Form1.LinearFunction.state:= cbUnchecked;
  Form1.QuadraticFunction.state:= cbUnchecked;
  Form1.CubicFunction.state:= cbUnchecked;
  Form1.SineFunction.state:= cbUnchecked;
  Form1.CosineFunction.state:= cbUnchecked;
  Form1.TangentFunction.state:= cbUnchecked;
  Form1.ExponentialFunction.state:= cbUnchecked;
  Form1.InverseFunction.state:= cbUnchecked;
end;

// Function Plotting Procedures

Procedure TLinear.PlotFunction(PlotPanel1: TPlotPanel);
Var
  i : integer;
Begin
  for i:= -10 to +10 do PlotPanel1.AddXY(i, i,clred,0);
end;

Procedure TQuadratic.PlotFunction(PlotPanel1: TPlotPanel);
Var
  i : integer;
Begin
  for i:= -1000 to +1000 do
    Begin
      YValue := (0.01*i)*(0.01*i);
      PlotPanel1.AddXY((0.01*i),YValue ,clred,0);
    end;
```

```
end;

Procedure TCubic.PlotFunction(PlotPanel1: TPlotPanel);
Var
 i : integer;
Begin
  for i:= -1000 to +1000 do
    Begin
     YValue := (0.01*i)*(0.01*i)*(0.01*i);
     PlotPanel1.AddXY((0.01*i),YValue ,clred,0);
    end;
end;

Procedure TSine.PlotFunction(PlotPanel1: TPlotPanel);
Var
 i : integer;
Begin
  for i:=-360 to 360 do
    begin
     YValue:=sin((PI/180)*i);
     PlotPanel1.AddXY(i,YValue,clred,0);
    end;
end;

Procedure TCosine.PlotFunction(PlotPanel1: TPlotPanel);
Var
 i : integer;
Begin
 for i:=-360 to 360 do
    begin
     YValue:=cos((PI/180)*i);
     PlotPanel1.AddXY(i,YValue,clRed,0);
    end;
end;

Procedure TTangent.PlotFunction(PlotPanel1: TPlotPanel);
Var
 i : integer;
Begin
for i:=-360 to 360 do
    begin
    if i mod 90 = 0
      then
       begin
        YValue:=(tan((PI/180)*(i-0.001)));
        PlotPanel1.AddXY(i,YValue,clred,0);
       end
      else
       begin
        YValue:=(tan((PI/180)*i));
        PlotPanel1.AddXY(i,YValue,clRed,0);
       end;
```

```
    end;
end;

Procedure TExponential.PlotFunction(PlotPanel1: TPlotPanel);
Var
 i : integer;
Begin
  for i:= -1000 to +300 do
    Begin
     YValue := Exp(0.01*i);
     PlotPanel1.AddXY((0.01*i),YValue ,clred,0);
    end;
end;

Procedure TInverse.PlotFunction(PlotPanel1: TPlotPanel);
Var
 i : integer;
Begin
  for i:= -1000 to -1 do
    Begin
     YValue := (1/(0.01*i));
     PlotPanel1.AddXY((0.01*i),YValue ,clred,0);
    end;
  for i:= 1 to 1000  do
    Begin
     YValue := (1/(0.01*i));
     PlotPanel1.AddXY((0.01*i),YValue ,clred,0);
    end;
end;

Procedure TModulus.PlotFunction(PlotPanel1: TPlotPanel);
Var
 i : integer;
Begin
  for i:= -1000 to 1000 do
    Begin
     YValue := (abs(0.01*i));                  //"abs(x)" returns the absolute
     PlotPanel1.AddXY((0.01*i),YValue ,clred,0);   // value of x
    end;
end;

//FunctionTransformingProcedures

Procedure TFunction.GetFunctionTransformationMagnitudes(Form1: TForm1);
Begin
 a:= Form1.YTranslation.Position;
 b:= Form1.XTranslation.Position;
 c:= Form1.Ystretch.Position;
 d:= Form1.XStretch.Position;
 e:= Form1.XCoordinate.Position;
 if Form1.ReflectX.State = cbChecked
  then f  := -1
```

```
    else f  := 1;
  if Form1.ReflectY.State = cbChecked
    then g := -1
    else g := 1;
  if Form1.InverseX.State = cbChecked
    then InvX := true
    else InvX := false;
end;


Procedure TLinear.ApplyTransformation(PlotPanel1: TPlotPanel);
Var
 i : integer;
Begin
  PlotPanel1.Freeze(True);
  PlotPanel1.HideLayer(3);
  for i:= -10 to +10 do
    Begin
     YValue:=f *g*((d*(c*(i+b)))+a);
     PlotPanel1.AddXY(i, YValue ,clblue,3);
    end;
  PlotPanel1.UnHideLayer(3);
  if InvX = true
    Then
      Begin
       PlotPanel1.HideLayer(0);
       PlotPanel1.HideLayer(3);
       PlotPanel1.HideLayer(4);
       for i:= -1000 to -1 do
         Begin
          YValue := (1/(0.01*i));
          PlotPanel1.AddXY((0.01*i),YValue ,clmaroon,4);
         end;
       for i:= 1 to 1000  do
         Begin
          YValue := (1/(0.01*i));
          PlotPanel1.AddXY((0.01*i),YValue ,clmaroon,4);
         end;
       PlotPanel1.UnHideLayer(4);
      end;
end;


Procedure TQuadratic.ApplyTransformation(PlotPanel1: TPlotPanel);
Var
 i : integer;
Begin
  PlotPanel1.Freeze(True);
  PlotPanel1.HideLayer(3);
  for i:= -1000 to +1000 do
    Begin
     YValue := f *(c*((g*(d*((0.01*i)+b)))*(g*(d*((0.01*i)+b))))+a);
```

```
      PlotPanel1.AddXY((0.01*i),YValue ,clblue,3);
    end;
  PlotPanel1.UnHideLayer(3);
  if InvX = true
   Then
    Begin
     PlotPanel1.HideLayer(0);
     PlotPanel1.HideLayer(3);
     PlotPanel1.HideLayer(4);
     for i:= -1000 to -1 do
      Begin
       YValue := 1/((0.01*i)*(0.01*i));
       PlotPanel1.AddXY((0.01*i),YValue ,clmaroon,4);
      end;
     for i:= 1 to 1000  do
      Begin
       YValue := 1/((0.01*i)*(0.01*i));
       PlotPanel1.AddXY((0.01*i),YValue ,clmaroon,4);
      end;
     PlotPanel1.UnHideLayer(4);
    end;
end;

Procedure TCubic.ApplyTransformation(PlotPanel1: TPlotPanel);
Var
 i : integer;
Begin
  PlotPanel1.Freeze(True);
  PlotPanel1.HideLayer(3);
  for i:= -1000 to +1000 do
   Begin
    YValue := f *(c*((g*(d*((0.01*i)+b)))*(g*(d*((0.01*i)+b)))*(g*(d*((0.01*i)+b))))+a);
    PlotPanel1.AddXY((0.01*i),YValue ,clblue,3);
   end;
  PlotPanel1.UnHideLayer(3);
  if InvX = true
   Then
    Begin
     PlotPanel1.HideLayer(0);
     PlotPanel1.HideLayer(3);
     PlotPanel1.HideLayer(4);
     for i:= -1000 to -1 do
      Begin
       YValue := 1/((0.01*i)*(0.01*i)*(0.01*i));
       PlotPanel1.AddXY((0.01*i),YValue ,clmaroon,4);
      end;
     for i:= 1 to 1000  do
      Begin
       YValue := 1/((0.01*i)*(0.01*i)*(0.01*i));
       PlotPanel1.AddXY((0.01*i),YValue ,clmaroon,4);
      end;
     PlotPanel1.UnHideLayer(4);
```

```
      end;
end;

Procedure TSine.ApplyTransformation(PlotPanel1: TPlotPanel);
Var
 i : integer;
Begin
  PlotPanel1.Freeze(True);
  PlotPanel1.HideLayer(3);
  for i:=-360 to 360 do
   begin
    YValue:=f *(c*(sin(g*(d*(((PI/180)*i)+b))))+a);
    PlotPanel1.AddXY(i,YValue,clblue,3);
   end;
  PlotPanel1.UnHideLayer(3);
  if InvX = true
   Then
    Begin
     PlotPanel1.HideLayer(0);
     PlotPanel1.HideLayer(3);
     PlotPanel1.HideLayer(4);
     for i:=-360 to 360 do
      begin
       if i mod 180 = 0
        then
         begin
          YValue:=(csc((PI/180)*(i-0.001)));    //csc(x)= 1/sin(x)
          PlotPanel1.AddXY(i,YValue,clmaroon,4);
         end
        else
         begin
          YValue:=(csc((PI/180)*i));
          PlotPanel1.AddXY(i,YValue,clmaroon,4);
         end;
      end;
     PlotPanel1.UnHideLayer(4);
    end;
end;

Procedure TCosine.ApplyTransformation(PlotPanel1: TPlotPanel);
Var
 i : integer;
Begin
  PlotPanel1.Freeze(True);
  PlotPanel1.HideLayer(3);
  for i:=-360 to 360 do
   begin
    YValue:=f *(c*(cos(g*(d*(((PI/180)*i)+b))))+a);
    PlotPanel1.AddXY(i,YValue,clblue,3);
   end;
  PlotPanel1.UnHideLayer(3);
  if InvX = true
```

```
    Then
     Begin
      PlotPanel1.HideLayer(0);
      PlotPanel1.HideLayer(3);
      PlotPanel1.HideLayer(4);
      for i:= -360 to 360 do
       begin
        if (i=-270) or (i=-90) or (i=90) or (i=270)
         then
          begin
           YValue:=(sec((PI/180)*(i-0.001)));    //sec(x)=1/cos(x)
           PlotPanel1.AddXY(i,YValue,clmaroon,4);
          end
         else
          begin
           YValue:=(sec((PI/180)*i));
           PlotPanel1.AddXY(i,YValue,clmaroon,4);
          end;
       end;
      PlotPanel1.UnHideLayer(4);
     end;
end;

Procedure TTangent.ApplyTransformation(PlotPanel1: TPlotPanel);
Var
 i : integer;
Begin
  PlotPanel1.Freeze(True);
  PlotPanel1.HideLayer(3);
  for i:=-360 to 360 do
   begin
   if i mod 90 = 0
     then
      begin
       YValue:=f *(c*(tan(g*(d*(((PI/180)*(i-0.001))+b))))+a);  //got a problem with "d"
       PlotPanel1.AddXY(i,YValue,clblue,3);
      end
     else
      begin
       YValue:=f *(c*(tan(g*(d*(((PI/180)*i)+b))))+a);
       PlotPanel1.AddXY(i,YValue,clblue,3);
      end;
   end;
  PlotPanel1.UnHideLayer(3);
  if InvX = true
   Then
    Begin
     PlotPanel1.HideLayer(0);
     PlotPanel1.HideLayer(3);
     PlotPanel1.HideLayer(4);
     for i:= -360 to 360 do
      begin
```

```
        if i mod 180 = 0
         then
          begin
           YValue:=(cot((PI/180)*(i-0.001)));    //cot(x)=1/tan(x)
           PlotPanel1.AddXY(i,YValue,clmaroon,4);
          end
         else
          begin
           YValue:=(cot((PI/180)*i));
           PlotPanel1.AddXY(i,YValue,clmaroon,4);
          end;
       end;
      PlotPanel1.UnHideLayer(4);
     end;
end;

Procedure TExponential.ApplyTransformation(PlotPanel1: TPlotPanel);
Var
 i : integer;
Begin
  PlotPanel1.Freeze(True);
  PlotPanel1.HideLayer(3);
  for i:= -1000 to +1000 do
   Begin
    YValue :=f *(c*(Exp(g*(d*(0.01*i)+b)))+a);          //Problem with "d" and "b" extremes
    PlotPanel1.AddXY((0.01*i),YValue ,clblue,3);    //try decreasing values maybe?
   end;
  PlotPanel1.UnHideLayer(3);
  if InvX = true
   Then
    Begin
     PlotPanel1.HideLayer(0);
     PlotPanel1.HideLayer(3);
     PlotPanel1.HideLayer(4);
     for i:= -1000 to 1000 do
      Begin
       YValue := 1/(exp(0.01*i));
       PlotPanel1.AddXY((0.01*i),YValue ,clMaroon,4);
      end;
     PlotPanel1.UnHideLayer(4);
    end;
end;

Procedure TInverse.ApplyTransformation(PlotPanel1: TPlotPanel);
Var
 d1 : extended;
 i : integer;
Begin
  PlotPanel1.Freeze(True);
  PlotPanel1.HideLayer(3);
  d1:= d;
  if d1 = 0
```

```
    then d1:= d1 + 0.01;   //error validation
  for i:= -1000 to (-1-(100*b)) do
   Begin
    YValue := f *(c*(1/(g*(d1*((0.01*i)+b))))+a);          //"b" due to changing assymptote (fixed)
    PlotPanel1.AddXY((0.01*i),YValue ,clblue,3);    //"d" due to zero
   end;
  for i:= (1-(100*b)) to 1000  do
   Begin
    YValue := f *(c*(1/(g*(d1*((0.01*i)+b))))+a);
    PlotPanel1.AddXY((0.01*i),YValue ,clblue,3);
   end;
  PlotPanel1.UnHideLayer(3);
  if InvX = true
   Then
    Begin
     PlotPanel1.HideLayer(0);
     PlotPanel1.HideLayer(3);
     PlotPanel1.HideLayer(4);
     for i:= -1000 to 1000 do
      Begin
       YValue :=(0.01*i);
       PlotPanel1.AddXY((0.01*i),YValue ,clmaroon,4);
      end;
     PlotPanel1.UnHideLayer(4);
    end;
end;

Procedure TModulus.ApplyTransformation(PlotPanel1: TPlotPanel);
Var
 i : integer;
Begin
  PlotPanel1.Freeze(True);
  PlotPanel1.HideLayer(3);
  for i:= -1000 to 1000 do
   Begin
    YValue := f *(c*(Abs(g*(d*(0.01*i)+b)))+a);
    PlotPanel1.AddXY((0.01*i),YValue ,clblue,3);
   end;
  PlotPanel1.UnHideLayer(3);
  if InvX = true
   Then
    Begin
     PlotPanel1.HideLayer(0);
     PlotPanel1.HideLayer(3);
     PlotPanel1.HideLayer(4);
     for i:= -1000 to -1 do
      Begin
       YValue :=1/(Abs(0.01*i));
       PlotPanel1.AddXY((0.01*i),YValue ,clmaroon,4);
      end;
     for i:= 1 to 1000 do
      Begin
```

```
        YValue :=1/(Abs(0.01*i));
        PlotPanel1.AddXY((0.01*i),YValue ,clmaroon,4);
       end;
      PlotPanel1.UnHideLayer(4);
    end;
end;

//HighlightPoints

Procedure TLinear.HighlightPoints(PlotPanel1: TPlotPanel);
Begin
 if InvX = false
  then
    Begin
     PlotPanel1.HideLayer(5);
     PlotPanel1.HideLayer(6);
     YValue:=f *g*((d*(c*(e+b)))+a);
     PlotPanel1.AddXY(e, YValue ,clblue,5);
     PlotPanel1.AddXY(e,e,clred,6);
     PlotPanel1.UnHideLayer(0);
     PlotPanel1.UnHideLayer(5);
     PlotPanel1.UnHideLayer(6);
     PlotPanel1.HideLayer(4);
    end;
  PlotPanel1.Freeze(false);
end;

Procedure TQuadratic.HighlightPoints(PlotPanel1: TPlotPanel);
Begin
 if InvX = false
  then
    Begin
     PlotPanel1.HideLayer(5);
     PlotPanel1.HideLayer(6);
     YValue := f *(c*((g*(d*((e)+b)))*(g*(d*((e)+b))))+a);
     PlotPanel1.AddXY(e, YValue ,clblue,5);
     PlotPanel1.AddXY(e,e*e,clred,6);
     PlotPanel1.UnHideLayer(0);
     PlotPanel1.UnHideLayer(5);
     PlotPanel1.UnHideLayer(6);
     PlotPanel1.HideLayer(4);
    end;
 PlotPanel1.Freeze(false);
end;

Procedure TCubic.HighlightPoints(PlotPanel1: TPlotPanel);
Begin
 if InvX = false
  then
    Begin
     PlotPanel1.HideLayer(5);
     PlotPanel1.HideLayer(6);
```

```
      YValue := f *(c*((g*(d*((e)+b)))*(g*(d*((e)+b)))*(g*(d*((e)+b))))+a);
      PlotPanel1.AddXY(e, YValue ,clblue,5);
      PlotPanel1.AddXY(e,e*e*e,clred,6);
      PlotPanel1.UnHideLayer(0);
      PlotPanel1.UnHideLayer(5);
      PlotPanel1.UnHideLayer(6);
      PlotPanel1.HideLayer(4);
     end;
 PlotPanel1.Freeze(false);
end;

Procedure TSine.HighlightPoints(PlotPanel1: TPlotPanel);
Begin
 if InvX = false
  then
    Begin
     PlotPanel1.HideLayer(5);
     PlotPanel1.HideLayer(6);
     YValue:= f *(c*(sin(g*(d*(((PI/180)*(e*72))+b))))+a);
     PlotPanel1.AddXY(e*72, YValue ,clblue,5);
     PlotPanel1.AddXY(e*72,sin((PI/180)*(e*72)),clred,6);
     PlotPanel1.UnHideLayer(0);
     PlotPanel1.UnHideLayer(5);
     PlotPanel1.UnHideLayer(6);
     PlotPanel1.HideLayer(4);
    end;
 PlotPanel1.Freeze(false);
end;

Procedure TCosine.HighlightPoints(PlotPanel1: TPlotPanel);
Begin
 if InvX = false
  then
    Begin
     PlotPanel1.HideLayer(5);
     PlotPanel1.HideLayer(6);
     YValue:= f *(c*(cos(g*(d*(((PI/180)*(e*72))+b))))+a);
     PlotPanel1.AddXY(e*72, YValue ,clblue,5);
     PlotPanel1.AddXY(e*72,cos((PI/180)*(e*72)),clred,6);
     PlotPanel1.UnHideLayer(0);
     PlotPanel1.UnHideLayer(5);
     PlotPanel1.UnHideLayer(6);
     PlotPanel1.HideLayer(4);
    end;
 PlotPanel1.Freeze(false);
end;

Procedure TTangent.HighlightPoints(PlotPanel1: TPlotPanel);
Begin
 if InvX = false
  then
    Begin
```

```
        PlotPanel1.HideLayer(5);
        PlotPanel1.HideLayer(6);
        YValue:= f *(c*(tan(g*(d*(((PI/180)*((e-0.001)*72))+b))))+a);
        PlotPanel1.AddXY(e*72, YValue ,clblue,5);
        PlotPanel1.AddXY(e*72,tan((PI/180)*((e-0.001)*72)),clred,6);
        PlotPanel1.UnHideLayer(0);
        PlotPanel1.UnHideLayer(5);
        PlotPanel1.UnHideLayer(6);
        PlotPanel1.HideLayer(4);
      end;
  PlotPanel1.Freeze(false);
end;

Procedure TExponential.HighlightPoints(PlotPanel1: TPlotPanel);
Begin
 if InvX = false
   then
     Begin
      PlotPanel1.HideLayer(5);
      PlotPanel1.HideLayer(6);
      YValue:= f *(c*(Exp(g*(d*(e)+b)))+a);
      PlotPanel1.AddXY(e, YValue ,clblue,5);
      PlotPanel1.AddXY(e,Exp(e),clred,6);
      PlotPanel1.UnHideLayer(0);
      PlotPanel1.UnHideLayer(5);
      PlotPanel1.UnHideLayer(6);
      PlotPanel1.HideLayer(4);
     end;
 PlotPanel1.Freeze(false);
end;

Procedure TInverse.HighlightPoints(PlotPanel1: TPlotPanel);
Var
 d1 : real;
Begin
 d1:= d;
 if d1 = 0
   then d1:= d1 + 0.01;
 if InvX = false
   then
     Begin
      PlotPanel1.HideLayer(5);
      PlotPanel1.HideLayer(6);
      YValue:= f *(c*(1/(g*(d1*((e-0.001)+b))))+a);
      PlotPanel1.AddXY(e, YValue ,clblue,5);
      PlotPanel1.AddXY(e,(1/(e-0.001)),clred,6);
      PlotPanel1.UnHideLayer(0);
      PlotPanel1.UnHideLayer(5);
      PlotPanel1.UnHideLayer(6);
      PlotPanel1.HideLayer(4);
     end;
 PlotPanel1.Freeze(false);
```

```
end;

Procedure TModulus.HighlightPoints(PlotPanel1: TPlotPanel);
Begin
 if InvX = false
  then
    Begin
     PlotPanel1.HideLayer(5);
     PlotPanel1.HideLayer(6);
     YValue := f *(c*(Abs(g*(d*(e)+b)))+a);
     PlotPanel1.AddXY(e, YValue ,clblue,5);
     PlotPanel1.AddXY(e,Abs(e),clred,6);
     PlotPanel1.UnHideLayer(0);
     PlotPanel1.UnHideLayer(3);
     PlotPanel1.UnHideLayer(5);
     PlotPanel1.UnHideLayer(6);
     PlotPanel1.HideLayer(4);
    end;
 PlotPanel1.Freeze(false);
end;

//Graph Information

Procedure TFunction.CalculateTransformedGraphInfo;
Begin
 Case a of
  1..10  : Sa:= '+'+IntToStr(a);
  0      : Sa:= '';
 else
  Sa:= IntToStr(a);
 end;
 Case b of
  1..10  : Sb:= '(x+'+IntToStr(b)+')';
  0      : Sb:= '(x)'
 else
  Sb:= '(x'+IntToStr(b)+')';
 end;
 Case c of
  2..10  : Sc := IntToStr(c);
  1      : Sc := '';
  0      : Sc := IntToStr(c);
 else
  Begin
   if c = -1
    then Sc := '-'+IntToStr(c)
    else Sc := IntToStr(c);
  end;
 end;
 Case d of
  2..10  : Sd := '('+IntToStr(d)+Sb+')';
  1      : Sd := Sb;
  0      : Sd := '(0)';
```

```
  else
   Begin
    if d = -1
     then Sd := '(-'+Sb+')'
     else Sd := '('+IntToStr(d)+Sb+')';
   end;
  end;
  if f = -1
   then
    Begin
    if g = -1
     then TextInfo := '-('+Sc+'f'+'(-'+Sd+')'+Sa+')'
     else TextInfo := '-('+Sc+'f'+Sd+Sa+')'
    end
   else
    Begin
     if g = -1
      then TextInfo := Sc+'f'+'(-'+Sd+')'+Sa
      else TextInfo := Sc+'f'+Sd+Sa
    end;
end;

Procedure TFunction.DisplayCartCoordAndTransFunctTxt(Form1: TForm1);
Begin
 if InvX = true
  then
   begin
    Form1.OrigFunctTxt.Font.Color:= clMaroon;
    Form1.TransFunctTxt.Caption:= '';
    Form1.OrigCartCoord.Caption:= '';
    Form1.TransCartCoord.Caption:= '';
   end
  else
   begin
    Form1.OrigFunctTxt.Font.Color:= clRed;
    Form1.TransFunctTxt.Caption:= TextInfo;
    XRed := IntToStr(e);
    XBlue := IntToStr(e);
    Form1.OrigCartCoord.Caption := '('+XRed+','+YRed+')';
    Form1.TransCartCoord.Caption := ' ('+XBlue+','+YBlue+')';
   end;
end;

Procedure TLinear.DisplayOrigTxtAndCalcCartCoord(Form1: TForm1);
Begin
 if InvX = true
  then Form1.OrigFunctTxt.Caption:= '1/f(x) = 1/x'
  else
   Begin
    Form1.OrigFunctTxt.Caption:= 'f(x) = x';
    YRed := IntToStr(e);
    YBlue := IntToStr(f *g*((d*(c*(e+b)))+a));
```

```
    end;
end;

Procedure TQuadratic.DisplayOrigTxtAndCalcCartCoord(Form1: TForm1);
Begin
 if InvX = true
   then  Form1.OrigFunctTxt.Caption:= '1/f(x) = 1/x²'
   else
    Begin
     Form1.OrigFunctTxt.Caption:= 'f(x) = x²';
     YRed := IntToStr(e*e);
     YBlue := IntToStr(f *(c*((g*(d*((e)+b)))*(g*(d*((e)+b))))+a));
    end;
end;

Procedure TCubic.DisplayOrigTxtAndCalcCartCoord(Form1: TForm1);
Begin
 if InvX = true
   then Form1.OrigFunctTxt.Caption:= '1/f(x) = 1/x³'
   else
    Begin
     Form1.OrigFunctTxt.Caption:= 'f(x) = x³';
     YRed := IntToStr(e*e*e);
     YBlue := IntToStr(f *(c*((g*(d*((e)+b)))*(g*(d*((e)+b)))*(g*(d*((e)+b))))+a));
    end;
end;

Procedure TSine.DisplayOrigTxtAndCalcCartCoord(Form1: TForm1);
Begin
 if InvX = true
   then Form1.OrigFunctTxt.Caption:= '1/f(x) = 1/sin(x)'
   else
    Begin
     Form1.OrigFunctTxt.Caption:= 'f(x) = sin(x)';
     YRed := FloatToStr(sin((PI/180)*(e*72)));
     YBlue := FloatToStr(f *(c*(sin(g*(d*(((PI/180)*(e*72))+b))))+a));
    end;
end;

Procedure TCosine.DisplayOrigTxtAndCalcCartCoord(Form1: TForm1);
Begin
 if InvX = true
   then Form1.OrigFunctTxt.Caption:= '1/f(x) = 1/cos(x)'
   else
    Begin
     Form1.OrigFunctTxt.Caption:= 'f(x) = cos(x)';
     YRed := FloatToStr(cos((PI/180)*(e*72)));
     YBlue := FloatToStr(f *(c*(cos(g*(d*(((PI/180)*(e*72))+b))))+a));
    end;
end;

Procedure TTangent.DisplayOrigTxtAndCalcCartCoord(Form1: TForm1);
```

```
Begin
 if InvX = true
  then Form1.OrigFunctTxt.Caption:= '1/f(x) = 1/tan(x)'
  else
   Begin
    Form1.OrigFunctTxt.Caption:= 'f(x) = tan(x)';
    YRed := FloatToStr (tan((PI/180)*((e-0.001)*72)));
    YBlue := FloatToStr(f *(c*(tan(g*(d*(((PI/180)*((e-0.001)*72))+b))))+a));
   end;
end;

Procedure TExponential.DisplayOrigTxtAndCalcCartCoord(Form1: TForm1);
Begin
 if InvX = true
  then Form1.OrigFunctTxt.Caption:= '1/f(x) = 1/(e^x)'
  else
   Begin
    Form1.OrigFunctTxt.Caption:= 'f(x) = e^x';
    YRed := FloatToStr(Exp(e));
    YBlue := FloatToStr( f *(c*(Exp(g*(d*(e)+b)))+a));
   end;
end;

Procedure TInverse.DisplayOrigTxtAndCalcCartCoord(Form1: TForm1);
Var
 d1 : real;
Begin
 if InvX = true
  then Form1.OrigFunctTxt.Caption:= '1/f(x) = x'
  else
   Begin
    d1:= d;
    if d1 = 0
     then d1:= d1 + 0.01;
    Form1.OrigFunctTxt.Caption:= 'f(x) = 1/x';
    YRed := FloatToStr (1/(e-0.001));
    YBlue := FloatToStr (f *(c*(1/(g*(d1*((e-0.001)+b))))+a));
   end;
end;

Procedure TModulus.DisplayOrigTxtAndCalcCartCoord(Form1: TForm1);
Begin
 if InvX = true
  then Form1.OrigFunctTxt.Caption:= 'f(x) = 1/|x|'
  else
   Begin
    Form1.OrigFunctTxt.Caption:= 'f(x) = |x|';
    YRed := IntToStr(Abs(e));
    YBlue := IntToStr(f *(c*(Abs(g*(d*(e)+b)))+a));
   end;
end;
```

```
{ TForm1 }

//The Functions

procedure TForm1.LinearFunctionClick(Sender: TObject);
begin
  Function1 := TLinear.Create;
  Function1.ToggleboxFix(Form1);
  Function1.SetXYAxesLimits(-10,10,-10,10,PlotPanel1);
  Function1.SetXYAxesIntervalsAndLayers(1,1,PlotPanel1);
  Function1.PlotXYAxes(PlotPanel1);
  Function1.PlotFunction(PlotPanel1);
  Function1.Free;
  UpdateTransformedGraph;

end;

procedure TForm1.ModulusFunctionClick(Sender: TObject);
begin
  Function1 := TModulus.Create;
  Function1.ToggleboxFix(Form1);
  Function1.SetXYAxesLimits(-10,10,-10,10,PlotPanel1);
  Function1.SetXYAxesIntervalsAndLayers(1,1,PlotPanel1);
  Function1.PlotXYAxes(PlotPanel1);
  Function1.PlotFunction(PlotPanel1);
  Function1.Free;
  UpdateTransformedGraph;
end;


procedure TForm1.CubicFunctionClick(Sender: TObject);
begin
  Function1 := TCubic.Create;
  Function1.ToggleboxFix(Form1);
  Function1.SetXYAxesLimits(-10,10,-10,10,PlotPanel1);
  Function1.SetXYAxesIntervalsAndLayers(1,1,PlotPanel1);
  Function1.PlotXYAxes(PlotPanel1);
  Function1.PlotFunction(PlotPanel1);
  Function1.Free;
  UpdateTransformedGraph;
end;

procedure TForm1.ExponentialFunctionClick(Sender: TObject);
begin
  Function1 := TExponential.Create;
  Function1.ToggleboxFix(Form1);
  Function1.SetXYAxesLimits(-10,10,-10,10,PlotPanel1);
  Function1.SetXYAxesIntervalsAndLayers(1,1,PlotPanel1);
  Function1.PlotXYAxes(PlotPanel1);
  Function1.PlotFunction(PlotPanel1);
  Function1.Free;
  UpdateTransformedGraph;
```

```
end;

procedure TForm1.InverseFunctionClick(Sender: TObject);
begin
  Function1 := TInverse.Create;
  Function1.ToggleboxFix(Form1);
  Function1.SetXYAxesLimits(-10,10,-10,10,PlotPanel1);
  Function1.SetXYAxesIntervalsAndLayers(1,1,PlotPanel1);
  Function1.PlotXYAxes(PlotPanel1);
  Function1.PlotFunction(PlotPanel1);
  Function1.Free;
  UpdateTransformedGraph;
end;

procedure TForm1.QuadraticFunctionClick(Sender: TObject);
begin
  Function1 := TQuadratic.Create;
  Function1.ToggleboxFix(Form1);
  Function1.SetXYAxesLimits(-10,10,-10,10,PlotPanel1);
  Function1.SetXYAxesIntervalsAndLayers(1,1,PlotPanel1);
  Function1.PlotXYAxes(PlotPanel1);
  Function1.PlotFunction(PlotPanel1);
  Function1.Free;
  UpdateTransformedGraph;
end;

procedure TForm1.SineFunctionClick(Sender: TObject);
begin
  Function1 := TSine.Create;
  Function1.ToggleboxFix(Form1);
  Function1.SetXYAxesLimits(-360,360,-5,5,PlotPanel1);
  Function1.SetXYAxesIntervalsAndLayers(90,1,PlotPanel1);
  Function1.PlotXYAxes(PlotPanel1);
  Function1.PlotFunction(PlotPanel1);
  Function1.Free;
  UpdateTransformedGraph;
end;

procedure TForm1.CosineFunctionClick(Sender: TObject);
Begin
  Function1 := TCosine.Create;
  Function1.ToggleboxFix(Form1);
  Function1.SetXYAxesLimits(-360,360,-5,5,PlotPanel1);
  Function1.SetXYAxesIntervalsAndLayers(90,1,PlotPanel1);
  Function1.PlotXYAxes(PlotPanel1);
  Function1.PlotFunction(PlotPanel1);
  Function1.Free;
  UpdateTransformedGraph;
end;

procedure TForm1.TangentFunctionClick(Sender: TObject);
Begin
```

```
  Function1 := TTangent.Create;
  Function1.ToggleboxFix(Form1);
  Function1.SetXYAxesLimits(-360,360,-5,5,PlotPanel1);
  Function1.SetXYAxesIntervalsAndLayers(90,1,PlotPanel1);
  Function1.PlotXYAxes(PlotPanel1);
  Function1.PlotFunction(PlotPanel1);
  Function1.Free;
  UpdateTransformedGraph;
end;


//The Function Transformations

procedure TForm1.XStretchChange(Sender: TObject);
Begin
 XStretchEdit.Value:= XStretch.Position;
 UpdateTransformedGraph;
end;


procedure TForm1.XTranslationChange(Sender: TObject);
Begin
 XTransEdit.Value:= XTranslation.Position;
 UpdateTransformedGraph;
end;

procedure TForm1.YstretchChange(Sender: TObject);
Begin
 YStretchEdit.Value:= YStretch.Position;
 UpdateTransformedGraph;
end;

procedure TForm1.YTranslationChange(Sender: TObject);
Begin
  YTransEdit.Value:= YTranslation.Position;
  UpdateTransformedGraph;
end;

procedure TForm1.XCoordinateChange(Sender: TObject);
begin
 XCoordEdit.Value:= XCoordinate.Position;
 UpdateTransformedGraph;
end;

procedure TForm1.ReflectXClick(Sender: TObject);
begin
 UpdateTransformedGraph;
end;

procedure TForm1.ReflectYClick(Sender: TObject);
begin
 UpdateTransformedGraph;
```

```
end;

procedure TForm1.InverseXClick(Sender: TObject);
begin
  PlotPanel1.Freeze(True);
  PlotPanel1.HideLayer(5);
  PlotPanel1.HideLayer(6);
  UpdateTransformedGraph;
  PlotPanel1.Freeze(False);
end;

//Extra Options

procedure TForm1.YTransEditChange(Sender: TObject);
begin
  YTranslation.Position:= YTransEdit.Value;
  UpdateTransformedGraph;
end;

procedure TForm1.XTransEditChange(Sender: TObject);
begin
  XTranslation.Position:= XTransEdit.Value;
  UpdateTransformedGraph;
end;

procedure TForm1.YStretchEditChange(Sender: TObject);
begin
   YStretch.Position:= YStretchEdit.Value;
   UpdateTransformedGraph;
end;

procedure TForm1.XStretchEditChange(Sender: TObject);
begin
   XStretch.Position:= XStretchEdit.Value;
   UpdateTransformedGraph;
end;

procedure TForm1.XCoordEditChange(Sender: TObject);
begin
   XCoordinate.Position:= XCoordEdit.Value;
   UpdateTransformedGraph;
end;

procedure TForm1.PrintButtonClick(Sender: TObject);

const
  TOTAL_PAGES = 1;     // How many pages to print
var
  printDialog    : TPrintDialog;
  page, startPage, endPage : Integer;
  MarginX, MarginY: longint;
```

```
begin
 // Create a printer selection dialog
 printDialog := TPrintDialog.Create(Form1);

 // Set up print dialog options
 printDialog.MinPage := 1;            // First allowed page number
 printDialog.MaxPage := TOTAL_PAGES;   // Highest allowed page number
 printDialog.ToPage  := TOTAL_PAGES;   // 1 to ToPage page range allowed
 printDialog.Options := [poPageNums];  // Allow page range selection

 // if the user has selected a printer (or default), then print!
 if printDialog.Execute then
  begin

   // Set the printjob title - as it it appears in the print job manager
   Printer.Title := 'Transformations of the graphs of functions';

   // Set the number of copies to print each page
   // This is crude - it doies not take Collation into account
   Printer.Copies := printDialog.Copies;

   //Set margins

   MarginX :=  optionsbox.PageSetupDialog.Margins.left;
   MarginY :=  optionsbox.PageSetupDialog.Margins.top;


   // Start printing
   Printer.BeginDoc;

   // Has the user selected a page range?
   if printDialog.PrintRange = prPageNums then
    begin
     startPage := printDialog.FromPage;
     endPage   := printDialog.ToPage;
    end
   else // All pages
    begin
     startPage := 1;
     endPage   := TOTAL_PAGES;
    end;

   // Set up the start page number
   page := startPage;

   // Show a message saying we are starting a page
   ShowMessagePos('Starting to print page '+IntToStr(page),300,300);

   // Allow Windows to keep processing messages
   Application.ProcessMessages;
   Printer.Canvas.Rectangle(optionsbox.PageSetupDialog.Margins);
   Printer.Canvas.Draw(MarginX, MarginY, plotpanel1.PlotBMP);
```

```
    // Finish printing
    Printer.EndDoc;
   end;
 end;

procedure TForm1.OptionsButtonClick(Sender: TObject);
begin
 Optionsbox.Visible:= true;
 Optionsbox.FormStyle:= fsStayOnTop;
end;

procedure TForm1.ResetButtonClick(Sender: TObject);
begin
 YTranslation.Position:= 0;
 XTranslation.Position:= 0;
 XStretch.Position:= 1;
 YStretch.Position:= 1;
 XCoordinate.Position:= 1;
 YTransEdit.Value:= 0;
 XTransEdit.Value:= 0;
 YStretchEdit.Value:= 1;
 XStretchEdit.Value:= 1;
 XCoordEdit.Value:= 1;
 ReflectX.State:= cbUnchecked;
 ReflectY.State:= cbUnchecked;
 InverseX.State:= cbUnchecked;
 UpdateTransformedGraph;
end;



initialization
 {$I unit1.lrs}



end.
```

OPTIONSBOX

```
unit Unit2;

{$mode objfpc}{$H+}

interface

uses
  Classes, SysUtils, FileUtil, LResources, Forms, Controls, Graphics, Dialogs,
  ButtonPanel, ExtCtrls, StdCtrls, PrintersDlgs;

type

  { TOptionsBox }

  TOptionsBox = class(TForm)
    PageSetup: TButton;
    CloseButn: TButton;
    XLabel: TCheckBox;
    YLabel: TCheckBox;
    GridLines: TCheckBox;
    GraphColour: TColorButton;
    PageSetupDialog: TPageSetupDialog;
    GraphSettings: TPanel;
    PrinterSettings: TPanel;
    Graph: TStaticText;
    Printer: TStaticText;
    procedure GraphColourColorChanged(Sender: TObject);
    procedure GridLinesClick(Sender: TObject);
    procedure CloseButnClick(Sender: TObject);
    procedure PageSetupClick(Sender: TObject);
    procedure XLabelClick(Sender: TObject);
    procedure YLabelClick(Sender: TObject);
  private
    { private declarations }
  public
    { public declarations }
  end;

var
  OptionsBox: TOptionsBox;

implementation

uses Unit1;

{ TOptionsBox }


procedure TOptionsBox.XLabelClick(Sender: TObject);
```

```
begin
  if XLabel.State = cbChecked
    then Form1.PlotPanel1.XMarks:= true
    else Form1.PlotPanel1.XMarks:= false;
end;

procedure TOptionsBox.YLabelClick(Sender: TObject);
begin
  if YLabel.State = cbChecked
    then Form1.PlotPanel1.YMarks:= true
    else Form1.PlotPanel1.YMarks:= false;
end;

procedure TOptionsBox.PageSetupClick(Sender: TObject);
begin
  PageSetupDialog.Execute = true;
end;

procedure TOptionsBox.GridLinesClick(Sender: TObject);
begin
  if GridLines.State = cbChecked
    then Form1.PlotPanel1.GridColor:= clDefault
    else Form1.PlotPanel1.GridColor:= Form1.PlotPanel1.BackColor;

end;

procedure TOptionsBox.CloseButnClick(Sender: TObject);
begin
  OptionsBox.Close;
end;

procedure TOptionsBox.GraphColourColorChanged(Sender: TObject);
begin
  Form1.PlotPanel1.BackColor:= GraphColour.ButtonColor;
end;

initialization
  {$I unit2.lrs}

end.
```

1.4 USER EVALUATION

Due to Time constraints this section couldn't be completed